Development Auxiliary Tool Guide (Training)

Issue 01
Date 2021-07-01
4.7.1.1 create_quant_config.................................................................153
4.7.1.2 quantize_model........................................................................157
4.7.1.3 save_model.............................................................................159
4.7.2 Quantization Aware Training.....................................................160
4.7.2.1 create_quant_retrain_config...................................................160
4.7.2.2 create_quant_retrain_model....................................................162
4.7.2.3 restore_quant_retrain_model..................................................163
4.7.2.4 save_quant_retrain_model......................................................165
4.8 FAQ..............................................................................................167
4.8.1 An Error Message Is Displayed During python3-tk Installation...............................................................................167
4.9 Appendixes...................................................................................167
5.1.2.1 Post-training Quantization and Quantization Aware Training..........................................................181
5.1.2.1.1 Terminology........................................................................181
5.1.2.1.2 Principles...........................................................................183
5.1.2.2 Accuracy-oriented Automatic Quantization Rollback...............185
5.1.2.3 Tensor Decomposition............................................................187
5.1.2.4 QAT Model to Ascend Model Conversion.................................189
5.1.2.5 Fusion Support.......................................................................190
5.1.3 Tool Workflow..........................................................................193
5.2 AMCT Installation..........................................................................195
5.2.1 Package Preparation.................................................................195
5.2.2 Pre-installation Actions.............................................................195
5.2.2.1 Ubuntu (x86)........................................................................196
5.2.3 Installation.................................................................................201
5.3 Post-training Quantization.............................................................201
5.3.1 Uniform Quantization.................................................................201
5.3.1.1 Quantizing an Image Classification Network.........................201
5.3.1.1.1 Quantization Preparations..................................................201
5.3.1.1.2 Quantization Example.......................................................202
5.3.1.2 Quantizing an Object Detection Network...............................204
5.3.1.3 Quantization Example Using the convert_model API..............205
5.3.2 Quantization Configuration.........................................................206
5.3.2.1 Overview..............................................................................207
5.3.2.2 Configuration File Options....................................................208
5.11.6 What Do I Do If My TensorFlow Network Output Node Is Changed by AMCT?.......................... 272
5.11.7 How Do I Restore the Model Training Parameters After Quantization Operators Are Inserted?.... 279
5.11.8 Python 3.7.5 Installation on Ubuntu.............................................................................................................280
6 AMCT Instructions (ONNX)........................................................................................................................... 282
   6.1 Introduction......................................................................................................................................................... 282
   6.1.1 Overview....................................................................................................................................................... 282
   6.1.2 Features............................................................................................................................................................ 283
   6.1.2.1 Post-training Quantization..............................................................................................................................283
   6.1.2.1.1 Terminology..................................................................................................................................................283
   6.1.2.1.2 Principles......................................................................................................................................................284
   6.1.2.2 QAT Model to Ascend Model Conversion....................................................................................................285
   6.1.3 Tool Workflow................................................................................................................................................285
   6.2 AMCT Installation.............................................................................................................................................. 287
   6.2.1 Package Preparation..................................................................................................................................... 287
   6.2.2 Pre-installation Actions.................................................................................................................................. 287
   6.2.2.1 Ubuntu (x86)..................................................................................................................................................287
   6.2.3 Installation..................................................................................................................................................... 290
   6.2.4 Post-installation Actions................................................................................................................................ 291
   6.2.4.1 Building and Installing Custom OPP..............................................................................................................291
   6.2.4.2 Setting Environment Variables......................................................................................................................291
   6.3 Post-training Quantization................................................................................................................................ 292
   6.3.1 Prerequisites................................................................................................................................................... 293
   6.3.2 Quantization Example...................................................................................................................................... 293
   6.3.3 Quantization Configuration.............................................................................................................................295
       6.3.3.1 Overview.....................................................................................................................................................295
       6.3.3.2 Configuration File Options..........................................................................................................................295
       6.3.3.3 Configuration Tuning................................................................................................................................295
   6.4 AMCT Update..................................................................................................................................................... 303
   6.5 AMCT Uninstallation.......................................................................................................................................... 304
   6.6 APIs..................................................................................................................................................................... 304
       6.6.1 create_quant_config......................................................................................................................................304
       6.6.2 quantize_model..............................................................................................................................................304
       6.6.3 save_model....................................................................................................................................................308
       6.6.4 convert_qat_model......................................................................................................................................309
   6.7 FAQ................................................................................................................................................................... 310
   6.7.1 An Error Message Is Displayed During python3-tk Installation.................................................................312
   6.8 Appendixes....................................................................................................................................................... 312
       6.8.1 Simplified Post-training Quantization Configuration File........................................................................312
       6.8.2 Quantization Factor Record File...................................................................................................................316
       6.8.3 Python 3.7.5 Installation on Ubuntu................................................................................................................320
7 AMCT Instructions (MindSpore)...................................................................................................................... 322
   7.1 Introduction....................................................................................................................................................... 322
8 AMCT Instructions (AscendCL) .......................................................... 385
8.1 Introduction.................................................................................. 385
8.1.1 Overview................................................................................ 385
8.1.2 Features................................................................................ 386
8.1.3 Tool Workflow......................................................................... 388
8.2 AMCT Installation........................................................................ 389
8.2.1 取软件包........................................................................... 389
8.2.2 Pre-installation Actions............................................................. 389
8.2.3 Installation............................................................................. 393
8.2.4 Post-installation Actions........................................................... 394
8.3 Quantization............................................................................... 394
8.4 Command-line Options................................................................. 396
8.4.1 Summary............................................................................... 397
8.4.2 Basic Functionality................................................................. 398
8.4.2.1 General Option................................................................. 398
8.4.2.1.1 --help or --h............................................................... 398
8.4.2.2 Input Options..................................................................... 398
8.4.2.2.1 --model....................................................................... 398
8.4.2.2.2 --weight...................................................................... 399
8.4.2.2.3 --framework............................................................... 399
8.4.2.2.4 --input_format............................................................ 400
8.4.2.2.5 --calibration_data....................................................... 401
8.4.2.2.6 --calibration_shape.................................................... 401
8.4.2.2.7 --calibration_config................................................. 402
8.4.2.3 Output Option.................................................................... 402
8.4.2.3.1 --output....................................................................... 402
8.4.2.4 Target SoC Option........................................................... 403
8.4.2.4.1 --soc_version......................................................... 403
8.4.3 Advanced Functionality........................................................ 404
8.4.3.1 Feature Options............................................................... 404
8.4.3.1.1 --out_nodes.............................................................. 404
8.4.3.1.2 --input_fp16_nodes.................................................. 405
8.4.3.2 Debug Option................................................................. 405
8.4.3.2.1 --log........................................................................ 405
8.5 AMCT Upgrade.......................................................................... 407
8.6 AMCT Uninstallation................................................................. 408
8.7 Appendixes.............................................................................. 408
8.7.1 Directory Structure of the Sample Package ................................................................................................................................ 408
8.7.2 Simplified Quantization Configuration File ................................................................................................................................ 409
8.7.3 Quantization Factor Record File .................................................................................................................................................. 413
8.7.4 Supported Command-line Options for AMCT Installation ........................................................................................................... 417

9 AMCT Instructions (TensorFlow, Ascend 910) ................................................................................................................................. 420
9.1 Introduction .................................................................................................................................................................................. 420
9.1.1 Overview .................................................................................................................................................................................. 420
9.1.2 Features .................................................................................................................................................................................. 421
9.1.2.1 Terminology .................................................................................................................................................................. 421
9.1.2.2 Implementation Principles ........................................................................................................................................ 422
9.1.3 Fusion Support ........................................................................................................................................................................ 423
9.1.4 Tool Workflow ........................................................................................................................................................................ 426
9.2 AMCT Installation ........................................................................................................................................................................ 427
9.2.1 获取软件包........................................................................................................................................................................ 427
9.2.2 Pre-installation Actions .......................................................................................................................................................... 427
9.2.3 Installation ............................................................................................................................................................................. 431
9.3 Post-training Quantization ............................................................................................................................................................. 432
9.3.1 Quantizing an Image Classification Network ...................................................................................................................... 432
9.3.1.1 Quantization Preparations ............................................................................................................................................. 432
9.3.1.2 Quantization Example ..................................................................................................................................................... 432
9.3.2 Quantizing an Object Detection Network ............................................................................................................................. 434
9.3.3 Quantization Configuration ..................................................................................................................................................... 436
9.3.3.1 Overview ...................................................................................................................................................................... 436
9.3.3.2 Configuration File Options ........................................................................................................................................ 437
9.3.3.3 Overview ......................................................................................................................................................................... 436
9.4 AMCT Update ............................................................................................................................................................................. 442
9.4.1 Tuning ..................................................................................................................................................................................... 442
9.5 AMCT Uninstallation ................................................................................................................................................................. 444
9.6 API Description ............................................................................................................................................................................. 445
9.6.1 Common APIs ........................................................................................................................................................................ 445
9.6.1.1 set_logging_level .......................................................................................................................................................... 445
9.6.2 Post-training Quantization ..................................................................................................................................................... 446
9.6.2.1 create_quant_config_ascend ......................................................................................................................................... 447
9.6.2.2 quantize_model_ascend ................................................................................................................................................. 449
9.6.2.3 save_model_ascend ......................................................................................................................................................... 451
9.6.2.4 create_quant_config ......................................................................................................................................................... 452
9.6.2.5 quantize_model ............................................................................................................................................................. 455
9.6.2.6 save_model .................................................................................................................................................................... 457
9.6.2.7 convert_model ............................................................................................................................................................. 458
9.6.3 File Options ............................................................................................................................................................................. 459
9.7 Appendixes ................................................................................................................................................................................ 459
9.7.1 Sample Package Directory Structure ................................................................................................................................. 460
9.7.2 Simplified Configuration File ............................................................................................................................................... 460
9.7.2.1 Simplified Configuration File Template (calibration_config_ascend_tf.proto) ......................................................... 460
10 Operator and Model Fast Query Instructions................................................. 474
10.1 Introduction....................................................................................................... 474
10.2 Instructions........................................................................................................ 475
10.3 Query Result...................................................................................................... 475

11 Profiling Instructions........................................................................................... 476
11.1 Introduction........................................................................................................ 476
11.1.1 Overview....................................................................................................... 476
11.1.2 Solution Description...................................................................................... 478
11.2 Restrictions........................................................................................................ 480
11.3 Profiling Workflow............................................................................................ 480
11.4 Environment Setup............................................................................................ 481
11.5 Collecting Profile Data..................................................................................... 481
11.5.1 Job Profiling.................................................................................................. 481
11.5.2 System Profiling............................................................................................ 483
11.6 Parsing Profile Data.......................................................................................... 486
11.6.1 Introduction to msprof.py.............................................................................. 487
11.6.2 Job Profiling.................................................................................................. 487
11.6.2.1 Parsing Profile Data.................................................................................. 487
11.6.2.2 Querying Training Job Details................................................................. 488
11.6.2.3 Obtaining Timeline Reports.................................................................... 489
11.6.2.3.1 Exporting Timeline Reports............................................................... 489
11.6.2.3.2 Degree of Parallelism of AI Core, AI CPU, and AllReduce.................... 491
11.6.2.3.3 Iteration Traces................................................................................... 492
11.6.2.4 Obtaining Summary Reports.................................................................... 493
11.6.2.4.1 Exporting Summary Reports.............................................................. 494
11.6.2.4.2 AI Core Summary............................................................................... 496
11.6.2.4.3 AI Core Operator Statistics................................................................. 498
11.6.2.4.4 Iteration Traces................................................................................... 498
11.6.2.4.5 Task Scheduler Summary................................................................. 500
11.6.2.4.6 AI CPU Summary.............................................................................. 501
11.6.2.4.7 Data Augmentation Summary............................................................ 501
11.6.2.4.8 Operator Fusion Summary................................................................. 502
11.6.3 System Profiling........................................................................................... 503
11.6.3.1 Parsing Profile Data in Any Directory..................................................... 503
11.6.3.2 Querying System Profile Data................................................................. 504
11.6.3.3 Obtaining Timeline Reports..................................................................... 505
11.6.3.3.1 Exporting Timeline Reports............................................................... 505
11.6.3.3.2 DDR Memory Read/Write Speed....................................................... 508
11.6.3.3.3 HBM Memory Read/Write Speed....................................................... 508
## 11.6.3.4 PCIe Bandwidth
509
## 11.6.3.5 NIC Summary
510
## 11.6.3.6 RoCE Bandwidth
510
## 11.6.3.7 L3 Cache Read/Write Speed
511
## 11.6.3.8 HCCS Bandwidth
512
## 11.6.3.9 Host CPU Utilization
512
## 11.6.3.10 Host Disk I/O Utilization
513
## 11.6.3.11 Host Memory Utilization
514
## 11.6.3.12 Host Network I/O Utilization
514
## 11.6.3.13 Host syscall and pthreadcall
515
## 11.6.3.4 Obtaining Summary Reports
515
## 11.6.3.4.1 Exporting Summary Reports
515
## 11.6.3.4.2 DDR Memory Read/Write Speed
519
## 11.6.3.4.3 HBM Memory Read/Write Speed
519
## 11.6.3.4.4 PCIe Bandwidth
520
## 11.6.3.4.5 AI CPU and Ctrl CPU Utilization
520
## 11.6.3.4.6 System Memory Utilization
521
## 11.6.3.4.7 AI Core Utilization
522
## 11.6.3.4.8 Process CPU Utilization
522
## 11.6.3.4.9 Process Memory Utilization
522
## 11.6.3.4.10 HCCS Bandwidth
523
## 11.6.3.4.11 NIC Summary
524
## 11.6.3.4.12 DVPP Summary
524
## 11.6.3.4.13 L3 Cache Read/Write Speed
525
## 11.6.3.4.14 RoCE Bandwidth
525
## 11.6.3.4.15 AI CPU Top Functions
526
## 11.6.3.4.16 AI CPU PMU Events
527
## 11.6.3.4.17 Ctrl CPU Top Functions
527
## 11.6.3.4.18 Ctrl CPU PMU Events
528
## 11.6.3.4.19 TS CPU Top Functions
528
## 11.6.3.4.20 TS CPU PMU Events
529
11.7 Analyzing Profile Data
529
## 11.7.1 Overview
529
## 11.7.2 Analyzing Profile Data Collected During MobileNetV2 Training
530
## 11.7.2.1 Analyzing Iteration Durations
530
## 11.7.2.2 Analyzing Operator Durations
531
## 11.7.3 Analyzing Profile Data Collected During AlexNet Training
532
## 11.7.3.1 Analyzing Iteration Durations
532
## 11.7.3.2 Analyzing Operator Durations
533
11.8 Appendix
534
## 11.8.1 Profiling Options
534
## 11.8.2 AI Core Metrics
535
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.8.3 Installing perf, iotop, and ltrace</td>
<td>537</td>
</tr>
<tr>
<td>11.8.4 FAQs</td>
<td>537</td>
</tr>
<tr>
<td>11.8.4.1 Profiling Fails Due to Full Disk Space</td>
<td>537</td>
</tr>
<tr>
<td><strong>12 Auto Tune Instructions</strong></td>
<td>539</td>
</tr>
<tr>
<td>12.1 About This Document</td>
<td>539</td>
</tr>
<tr>
<td>12.2 Introduction to Auto Tune</td>
<td>539</td>
</tr>
<tr>
<td>12.2.1 Overview</td>
<td>539</td>
</tr>
<tr>
<td>12.2.2 Architecture</td>
<td>541</td>
</tr>
<tr>
<td>12.2.3 Tuning Workflow</td>
<td>543</td>
</tr>
<tr>
<td>12.3 Tool Usage</td>
<td>545</td>
</tr>
<tr>
<td>12.3.1 Auto Tuning in Training Scenario</td>
<td>545</td>
</tr>
<tr>
<td>12.3.1.1 Environment Setup</td>
<td>545</td>
</tr>
<tr>
<td>12.3.1.2 Environment Variable Configuration</td>
<td>546</td>
</tr>
<tr>
<td>12.3.1.3 Tuning Procedure</td>
<td>551</td>
</tr>
<tr>
<td>12.3.1.4 Tuning Result</td>
<td>554</td>
</tr>
<tr>
<td>12.3.2 Repository Merging</td>
<td>556</td>
</tr>
<tr>
<td>12.3.3 Repository Merging</td>
<td>557</td>
</tr>
<tr>
<td>12.3.4 Troubleshooting</td>
<td>559</td>
</tr>
<tr>
<td>12.3.5 Operator Lists</td>
<td>559</td>
</tr>
<tr>
<td><strong>13 Model Accuracy Analyzer Instructions</strong></td>
<td>562</td>
</tr>
<tr>
<td>13.1 Overview</td>
<td>562</td>
</tr>
<tr>
<td>13.2 Data Preparation</td>
<td>563</td>
</tr>
<tr>
<td>13.2.1 Preparing .npy Data of a TensorFlow Network on GPU</td>
<td>563</td>
</tr>
<tr>
<td>13.2.2 Preparing Dump Data and Computational Graphs of a Model on the Ascend AI Processor</td>
<td>566</td>
</tr>
<tr>
<td>13.3 Vector Comparison</td>
<td>571</td>
</tr>
<tr>
<td>13.3.1 Network-wide Comparison</td>
<td>571</td>
</tr>
<tr>
<td>13.3.2 Single-Operator Comparison</td>
<td>574</td>
</tr>
<tr>
<td>13.4 Appendixes</td>
<td>578</td>
</tr>
<tr>
<td>13.4.1 How Do I Convert the Format of a Dump File?</td>
<td>578</td>
</tr>
<tr>
<td>13.4.2 How Do I View a Dump File?</td>
<td>581</td>
</tr>
<tr>
<td>13.4.3 How Do I Handle Exceptions in the Generated .npy File Names in Batches?</td>
<td>582</td>
</tr>
<tr>
<td>13.4.4 How Do I Migrate a Dump File from Windows to Linux?</td>
<td>584</td>
</tr>
<tr>
<td>13.4.5 compare_vector.py</td>
<td>585</td>
</tr>
<tr>
<td>13.4.5.1 Instructions</td>
<td>585</td>
</tr>
<tr>
<td>13.4.5.2 Network-wide Comparison</td>
<td>585</td>
</tr>
<tr>
<td>13.4.5.3 Single-Operator Comparison</td>
<td>588</td>
</tr>
<tr>
<td>13.4.5.4 How Do I Convert the Format of a Dump File?</td>
<td>591</td>
</tr>
<tr>
<td>13.4.5.5 How Do I View a Dump File?</td>
<td>593</td>
</tr>
<tr>
<td>13.4.5.6 How Do I Convert an .npy File into a Dump File?</td>
<td>593</td>
</tr>
<tr>
<td><strong>14 AI Core Error Analyzer Instructions</strong></td>
<td>595</td>
</tr>
<tr>
<td>14.1 Introduction</td>
<td>595</td>
</tr>
</tbody>
</table>
14.2 Troubleshooting Workflow ................................................................. 596
14.3 Preparing Data .................................................................................. 596
14.4 Analyzing AI Core Errors ................................................................. 598
14.5 Locating AI Core Errors ................................................................... 599
# 1 Document Description

<table>
<thead>
<tr>
<th>Document</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script Conversion Tool Instructions</td>
<td>GPU-based training and online inference scripts cannot be directly migrated to NPU. This document describes a tool dedicated to solving this problem through script conversion.</td>
<td>2 Script Conversion Tool Instructions</td>
</tr>
<tr>
<td>Profiling Instructions</td>
<td>This document describes how to use the Profiling tool. The tool is used to analyze key performance bottlenecks in the training phase and provide specific suggestions to achieve optimal product performance.</td>
<td>11 Profiling Instructions</td>
</tr>
<tr>
<td>Auto Tune Instructions</td>
<td>In training scenarios, the operator search tool can automatically optimize operators during network model compilation.</td>
<td>12 Auto Tune Instructions</td>
</tr>
<tr>
<td>Model Accuracy Analyzer Instructions</td>
<td>Huawei supports the migration of original networks for training on Ascend 910 AI Processor. As a result, the compute results of Huawei proprietary operators may be different from those of third-party equivalents (for example, from TensorFlow). The Model Accuracy Analyzer is designed to compare the compute results for developers to quickly resolve operator accuracy issues.</td>
<td>13 Model Accuracy Analyzer Instructions</td>
</tr>
<tr>
<td>Document</td>
<td>Description</td>
<td>Link</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>AI Core Error Analyzer Instructions</td>
<td>When you encounter AI Core errors in the training process, you can use the AI Core Error Analyzer to collect necessary information for quickly locating the AI Core errors.</td>
<td>14 AI Core Error Analyzer Instructions</td>
</tr>
</tbody>
</table>
2 Script Conversion Tool Instructions

2.1 Introduction

Overview

Many training and online inference scripts are executed on GPU instead of NPU, while NPU will be the dominant processor for AI compute capability. Due to the architecture differences between NPU and GPU, GPU-based training and online inference scripts cannot be directly used on NPU before being converted into scripts that support NPU.

NOTE

- msFmkTransplt provides suggestions and converts scripts by the adaptation rules, significantly accelerating script migration and reducing development workload. Note that the conversion result is for reference only.
- Currently, msFmkTransplt only supports the conversion of PyTorch training scripts.

System Requirement

msFmkTransplt runs on Ubuntu 18.04, CentOS 7.6, and EulerOS 2.8 only.

Environment Setup

Set up the development environment by referring to CANN Software Installation Guide.
2.2 Instructions

Command-line Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Value</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i</td>
<td>Sets the path of the original script to be converted.</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>--input</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-fmk</td>
<td>Sets the original script framework type. Currently, only conversion of 4 (PyTorch) is supported.</td>
<td>0: Caffe 1: MindSpore 3: TensorFlow 4: PyTorch</td>
<td>Yes</td>
</tr>
<tr>
<td>--framework</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-m</td>
<td>Sets the application scenario. Currently, only 0 (training) is supported.</td>
<td>0 (default): training 1: inference</td>
<td>No</td>
</tr>
<tr>
<td>--mode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-out</td>
<td>Sets the path for storing the converted script. Defaults to the original script path. The output directory for conversion result is saved as: <code>{original folder name}_msft</code></td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>--output</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-h</td>
<td>Displays help information.</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>--help</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2-1 Command-line options**

**Step 1**
Go to the directory of the script conversion tool msFmkTransplt.
```
cd {Ascend-CANN-Toolkit install path}/ascend-toolkit/{version}/{arch}-linux/toolkit/tools/ms_fmk_transplt
```

**Step 2**
Execute msFmkTransplt.
```
python3.7.5 ms_fmk_transplt.py -i {original script path} -fmk 4 -m 0 -out {conversion result path}
```

**Step 3**
Find the converted script in the specified output path.

----End

2.3 Result Analysis

You can view the result files in the output path when the script is converted.
```
xxx_msft  // Directory for storing script conversion results. The default directory is the directory of the original script. xxx indicates the name of the folder where the original script is stored.
```
## 2 Script Conversion Tool Instructions

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_reports</td>
<td>Directory of conversion result analysis reports</td>
</tr>
<tr>
<td>orig_script</td>
<td>Directory of the original script</td>
</tr>
<tr>
<td>Original script</td>
<td></td>
</tr>
<tr>
<td>msFmkTranspltResult.json</td>
<td>Conversion result analysis report</td>
</tr>
<tr>
<td>out_todo</td>
<td>To-do comment result directory. You can modify the script based on the comments to complete script conversion.</td>
</tr>
<tr>
<td>Script file with to-do comments</td>
<td></td>
</tr>
<tr>
<td>msFmkTranspltlog.txt</td>
<td>Log file generated during script conversion</td>
</tr>
</tbody>
</table>
3 AMCT Instructions (Caffe)

3.1 Introduction

3.1.1 Overview

This document describes how to quantize a Caffe model using Ascend Model Compression Toolkit (AMCT). In the quantization process, the precision of model weights and activations is reduced to make the model more compact, improving the compute efficiency while lowering the transfer latency.

AMCT is a Caffe-based Python toolkit that implements operator fusion (mainly BN fusion) and 8-bit quantization of activations and weights in neural networks. This toolkit decouples model quantization from model conversion. It implements independent quantization of quantization-capable operators in a model, and outputs a .prototxt model file and a .caffemodel weight file. The obtained accuracy simulation model can run on CPU or GPU to complete accuracy simulation. The obtained deployable model can run on the Ascend AI Processor with improved inference performance. This tool has the following advantages:
3.1.2 Features

3.1.2.1 Post-training Quantization and Quantization Aware Training

3.1.2.1.1 Terminology

There are two forms of quantization: post-training quantization and quantization aware training.

The foregoing two quantization forms are classified into weight quantization and activation quantization according to the quantization object, and are classified into uniform quantization and non-uniform quantization according to whether weights are compressed. Quantization aware training allows only uniform quantization.

As used in this document, the following terms have the meanings specified below.

Post-training Quantization

Post-training quantization refers to quantizing the weights of an already-trained model from float32 to int8 and calibrating and quantizing the activations by using a small calibration dataset. For details about the quantization workflow, see 3.3
Post-training Quantization. It is not supported to run a post-training quantization on more than one GPU.

- **Calibration dataset**
  During the calibration process, the algorithm uses each piece of data in the calibration dataset as input, accumulates the input data of each layer (or operator) that needs to be quantized, and uses the accumulated data as input data of the quantization algorithm to determine the quantization factors. Because the quantization factors and the accuracy of the quantized model are closely related to the selection of the calibration dataset, you are advised to calibrate the model using a subset of images from the validation dataset.

- **Activation quantization**
  Activation quantization is to collect statistics on the input data of each layer (or operator) to be quantized, to find the optimal pair of scale and offset per layer (or operator). For details, see [3.11.4 Quantization Factor Record File](#). Activations are the intermediate results of model inference computation. The value ranges are input-specific. Therefore, a group of reference inputs (a calibration dataset) needs to be used as incentives to record the input data of each layer (or operator) to be quantized for searching for quantization factors (scale and offset). During data calibration, extra memory (video memory/RAM) is needed to store the input data used to determine the quantization factors. Therefore, the video memory/RAM usage is higher than that required for performing inference only. The size of the extra memory is positively correlated with \( \text{batch\_size} \times \text{batch\_num} \) during calibration.

- **Weight quantization**
  After model training, the weights and the value ranges are determined. Therefore, quantization may be directly performed based on the value range of each weight.

Based on whether the weight data is compressed, quantization is classified into uniform quantization and non-uniform quantization. However, only uniform quantization is supported.

**Uniform quantization**: The quantized data is evenly distributed in a numerical space. For example, int8 quantization uses 8-bit int8 data to represent 32-bit fp32 data, and converts an fp32 operation (multiply-add operation) into an int8 operation, accelerating computing with reduced model size. In uniform int8 quantization, the quantized data is evenly distributed in the value range \([-128, +127]\) of int8. For details about the quantization workflow, see [3.3.2 Uniform Quantization](#).

If the accuracy drops significantly after uniform quantization, you need to retrain the model by referring to Quantization Aware Training or [3.1.2.2 Accuracy-oriented Automatic Quantization Rollback](#). The layers that support post-training quantization are listed as follows.
### Table 3-1 Layers that support post-training quantization and restrictions

<table>
<thead>
<tr>
<th>Technique</th>
<th>Layer</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform quantization</td>
<td>InnerProduct</td>
<td>transpose = false, axis = 1</td>
</tr>
<tr>
<td></td>
<td>Convolution</td>
<td>Using a 4 x 4 filter</td>
</tr>
<tr>
<td></td>
<td>Deconvolution</td>
<td>Using a 1-dilated 4 x 4 filter with group = 1</td>
</tr>
<tr>
<td></td>
<td>AVE Pooling</td>
<td>Global Pooling is not supported.</td>
</tr>
</tbody>
</table>

### Quantization Aware Training

Quantization aware training introduces quantization in the forward pass of the training process, allowing for higher accuracy.

Quantization aware training is time consuming and data hungry. For details about the quantization workflow, see 3.4 Quantization Aware Training.

- **Training dataset**
  Dataset of the already-trained network.

- **Activation quantization**
  Activation quantization refers to iterative training of the upper clip limit and lower clip limit, which are used to calculate the current scale and offset. The activation is the intermediate result of model inference and calculation. The ULQ retrain algorithm is used to continuously optimize the two factors during the quantization aware training process to obtain the optimal factors.

- **Weight quantization**
  Weight quantization means to optimize the quantization parameters of weights during the quantization aware training process to obtain the optimal parameters.

Quantization aware training allows only uniform quantization. The supported layers are listed as follows.

### Table 3-2 Layers that support quantization aware training and restrictions

<table>
<thead>
<tr>
<th>Layer</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>InnerProduct</td>
<td>transpose = false, axis = 1</td>
</tr>
<tr>
<td>Convolution</td>
<td>Using a 4 x 4 filter</td>
</tr>
<tr>
<td>Deconvolution</td>
<td>Using a 1-dilated 4 x 4 filter with group = 1</td>
</tr>
<tr>
<td>AVE Pooling</td>
<td>Global Pooling is not supported.</td>
</tr>
</tbody>
</table>
3.1.2.1.2 Implementation Principles

Figure 3-2 shows the AMCT principles. The operations in blue are implemented by users, and the operations in gray are implemented by using the AMCT API calls. You can import the library to the original Caffe network inference code and use the API calls at specific locations to implement the quantization function. The tool can be used in the following scenarios:

- **Post-training Quantization**
  - **Scenario 1**
    i. Construct an original Caffe model and then generate a quantization configuration file by using the 3.9.2.1 create_quant_config call.
    ii. Based on the Caffe model and quantization configuration file, initialize the tool by the 3.9.2.2 init call, configure the quantization factor record file, and parse the model into a graph.
    iii. Optimize the graph of the original Caffe model by using the 3.9.2.3 quantize_model call. The optimized model contains the quantization algorithm. Perform inference with the optimized model in the Caffe environment based on the image dataset and calibration dataset preset in AMCT to obtain the quantization factors. The image dataset is used to analyze the accuracy of the quantized data in the Caffe environment. The calibration dataset is used to generate quantization factors to ensure accuracy.
    iv. Save the quantized model by using the 3.9.2.4 save_model call to a model (including its weight file) for accuracy simulation in the Caffe environment or a model (including its weight file) deployable on the Ascend AI Processor.
  - **Scenario 2**
    Instead of using the APIs in scenario 1, if you have generated a quantized model based on your own quantization factors and original Caffe model, complete the quantization by using the 3.9.2.5 convert_model call. 3.3.2.3 Quantization Example Using the convert_model API gives a quantization example in this scenario.

- **Quantization Aware Training**
  a. Construct an original Caffe model and then generate a quantization configuration file by using the 3.9.3.1 create_quant_retrain_config call.
  b. Add the test phase (test_interval > 0, test_iter > 0) to solver.prototxt to enable the search of the shift factor N in the test phase and disable the precheck (test_initialization=false) to avoid false enablement of the search of the shift factor N. For details, see 3. (Note: Set net in the solver.prototxt file to the model generated by AMCT instead of train_net or test_net.)
  c. Optimize the original Caffe model by using the 3.9.3.2 create_quant_retrain_model call. The optimized model contains the quantization algorithm. Retrain the optimized model in the Caffe environment based on the image dataset and calibration dataset preset in AMCT to obtain the quantization factors.
  d. Save the quantized model by using the 3.9.3.3 save_quant_retrain_model call to a model (including its weight file) for
accuracy simulation in the Caffe environment or a model (including its weight file) deployable on the Ascend AI Processor.

**Figure 3-2 Tool principles**

Accuracy-oriented automatic quantization rollback is a technique introduced to ensure model accuracy, which automatically searches for the model quantization configuration and executes the post-training quantization process without sacrificing accuracy.

Accuracy-oriented automatic quantization rollback is similar to [3.3 Post-training Quantization](#). However, you do not need to manually tune the quantization configuration file, which greatly simplifies the optimization workload and improves the quantization efficiency. **Figure 3-3** shows the working principles.

**Figure 3-3 Principles of automatic quantization rollback**

The workflow is described as follows:

1. Generate a quantization configuration file by using the [3.9.2.1 create_quant_config](#) call, and then perform accuracy-oriented automatic quantization rollback by using the [3.9.2.6 accuracy_based_auto_calibration](#) call.

2. Pass the evaluator instance to the [3.9.2.6 accuracy_based_auto_calibration](#) call to analyze the accuracy of the original model.

In this process, the quantization strategy module in [3.9.2.6 accuracy_based_auto_calibration](#) is called to output the initialized quantization configuration file. The file records all layers that support quantization.
3. Run post-training quantization on the model based on the initial quantization configuration file (generated by the 3.9.2.1 create_quant_config call in 1) to obtain the accuracy of the fake-quantized model.

4. Compare the accuracy results of both models. If the fake-quantized model's degradation in accuracy is within acceptable limits, the quantized model is output. Otherwise, perform quantization rollback.
   a. Run inference on the original Caffe model and dump the input activations of each layer.
   b. Use the quantization factors obtained after post-training quantization to construct single-operator networks of quantization layers. Then, use the buffered activations to calculate the cosine similarity between the output data of each fake-quantized single-operator network and that of the original Caffe equivalent.
   c. Pass the cosine similarity list to the quantization strategy module in 3.9.2.6 accuracy_based_auto_calibration. The strategy module outputs a new quantization configuration file after certain layers are rolled back based on the initial quantization configuration file generated in 2.
   d. Run post-training quantization based on the new quantization configuration file to obtain a new fake-quantized model.
   e. Analyze the accuracy of the new fake-quantized model by a call to the evaluator module in 3.9.2.6 accuracy_based_auto_calibration.
      - If the model accuracy is acceptable, output a fake-quantized model and a deployable model.
      - If the model accuracy is unacceptable, the layer with the worst cosine similarity is rolled back, and go back to 4.c to output a new quantization configuration.

Figure 3-4 shows the rollback workflow.
3.1.2.3 Tensor Decomposition

Tensor decomposition converts a convolutional kernel into a stack of two smaller convolutional kernels to reduce the inference overhead. If the user model involves many convolution computations and most of the convolutional kernels have shapes larger than (64, 64, 3, 3), tensor decomposition is recommended. Otherwise, skip this step and proceed to quantization. Currently, tensor decomposition is supported under the following conditions:

- \( \text{group} = 1 \), \( \text{dilation} = (1,1) \), \( \text{stride} < 3 \)
- \( \text{kernel}_h = \text{kernel}_w \), \( \text{kernel}_h > 2 \)

Only when the original Caffe model has the Convolution layer and the layer meets the preceding conditions, the Convolution layer can be decomposed into two...
smaller Convolution layers. Then, you can use AMCT to convert the original Caffe model into a quantizable model that can be deployed on the Ascend AI Processor for better inference performance.

**Figure 3-5** shows the decomposition principle. Determine whether to decompose the original model as needed. For the tensor decomposition details, see 3.6 Tensor Decomposition.

**Figure 3-5 Tensor decomposition principle**

The procedure is as follows:

1. Call 3.9.4.1 auto_decomposition to perform tensor decomposition on the original model, resulting a new model file and a new weight file.
2. Fine-tune the decomposed model. Quantize the fine-tuned model by referring to 3.3 Post-training Quantization or 3.4 Quantization Aware Training.

**Figure 3-6** shows the ResNet-50 model before and after decomposition.
3.1.2.4 Fusion Support

Currently, this tool mainly implements the following forms of BN fusion:

- Convolution+BatchNorm+Scale fusion: Before quantization, the "Convolution +BatchNorm+Scale" composite in the model is fused into "Conv+BN+Scale." The BatchNorm and Scale layers are removed.

- Deconv+BN+Scale fusion: Before quantization, the "Deconvolution +BatchNorm+Scale" composite in the model is fused into "Deconv+BN+Scale." The BatchNorm and Scale layers are removed.
- BN+Scale+Conv fusion in post-training quantization: Before quantization, the "BatchNorm+Scale+Convolution" composite in the model is fused into "BN+Scale+Conv." The BatchNorm and Scale layers are removed.

- FC+BN+Scale fusion in post-training quantization: Before quantization, the "InnerProduct+BatchNorm+Scale" composite in the model is fused into "FC+BN+Scale." The BatchNorm and Scale layers are removed.

### 3.1.3 Tool Workflow

*Figure 3-7* shows the tool workflow.

**Table 3-3** Major actions in the tool workflow

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package preparation</td>
<td>Obtain the tool package by referring to <strong>3.2.1 Package Preparation</strong>.</td>
</tr>
</tbody>
</table>
**3.2 AMCT Installation**

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-installation actions</td>
<td>Before AMCT installation, create an installation user, check the system environment, install dependencies, and upload the AMCT package. For details, see <a href="#">3.2.2 Pre-installation Actions</a>.</td>
</tr>
<tr>
<td>Installation</td>
<td>Install the Caffe version AMCT by referring to <a href="#">3.2 AMCT Installation</a>.</td>
</tr>
<tr>
<td>Post-installation actions</td>
<td>After AMCT installation is complete, merge .proto files and install patch by referring to <a href="#">3.2.4 Post-installation Actions</a>, and then recompile the Caffe environment. To set the quantization logging level, you need to set related environment variables.</td>
</tr>
<tr>
<td>(Optional) Script creation with AMCT API calls</td>
<td>If you need to quantize your network model instead of the sample model provided in this instruction, you need to modify the script for adaptation before quantization. For details about how to parse the sample code, see <a href="#">3.3.1 Sample Code</a>.</td>
</tr>
<tr>
<td>Tensor decomposition</td>
<td>If the user model involves many convolution computations and most of the convolutional kernels have shapes larger than (64, 64, 3, 3), tensor decomposition is recommended. Otherwise, skip this step and proceed to quantization. For details about the tensor decomposition procedure, see <a href="#">3.6 Tensor Decomposition</a>.</td>
</tr>
<tr>
<td>Quantization</td>
<td>Run the provided quantization script to quantize an original network with the prepared dataset. There are two forms of quantization: post-training quantization and quantization aware training. For details, see <a href="#">3.3 Post-training Quantization</a> and <a href="#">3.4 Quantization Aware Training</a>. Post-training quantization can be further classified into uniform quantization and non-uniform quantization according to whether the weight data is compressed. This version supports only uniform quantization.</td>
</tr>
<tr>
<td>Automatic quantization rollback</td>
<td>Check the accuracy of the quantized model. If the accuracy of the quantized model is not satisfactory, perform <a href="#">3.5 Accuracy-oriented Automatic Quantization Rollback</a> or <a href="#">3.4 Quantization Aware Training</a>.</td>
</tr>
<tr>
<td>(Optional) Model conversion using ATC</td>
<td>You can convert the quantized deployable model to an offline model supported by the Ascend AI Processor by using ATC, and then perform subsequent inference.</td>
</tr>
</tbody>
</table>

---

(Caffe) Development Auxiliary Tool Guide (Training) Issue 01 (2021-07-01) Copyright © Huawei Technologies Co., Ltd.
3.2.1 Package Preparation

Currently, AMCT runs only on Ubuntu 18.04 (x86_64) servers. Before installation, click here to obtain the AMCT package Ascend-cann-amct_{software version}_ubuntu18.04-x86_64.tar.gz.

Before installation, obtain the AMCT package. AMCT runs on Ubuntu 18.04 (x86_64) or EulerOS (AArch64) servers. Select a required software package.

- Ubuntu 18.04 (x86_64) server:
  Ascend-amct-{software version}-ubuntu18.04.x86_64.tar.gz
- EulerOS (AArch64) server:
  Ascend-amct-{software version}-euleros2.9.aarch64.tar.gz

{software version} indicates the version number.

3.2.2 Pre-installation Actions

3.2.2.1 Ubuntu (x86)

Preparing the AMCT User

Any user (root or non-root) is allowed to install AMCT. This section uses a non-root user as an example.

- To install AMCT as the root user, skip this section.
- To install AMCT as an existing non-root user, ensure that the user has the read, write, and execute permissions on the $HOME directory.
- To install AMCT as a new non-root user, perform the following steps as the root user. The following uses this scenario as an example.
  a. Create an AMCT installation user and set the $HOME directory for the user:
     ```bash
     useradd -d /home/username -m username
     ```
  b. Set the user password:
     ```bash
     passwd username
     ```

NOTE

username indicates the name of the AMCT installation user. The umask value of the user is at least 0027.

- You can view the umask value by running the umask command.
- You can change the umask value by running the umask NewValue command.

(Optional) Setting the Permission of the AMCT Installation User

Skip this part if you install AMCT as the root user.

Before installing AMCT, you need to download the dependencies, which requires the sudo apt-get permission. Run the following commands as the root user:

1. Open the /etc/sudoers file:
   ```bash
   chmod u+w /etc/sudoers
   vi /etc/sudoers
   ```
2. Add the following content under **# User privilege specification** in the file:

```
username ALL=(ALL:ALL)   NOPASSWD:SETENV:/usr/bin/apt-get,/usr/bin/pip, /bin/tar, /bin/mkdir, /bin/sh,/bin/bash, /usr/bin/make, /usr/bin/pip3, /usr/bin/pip3.7, /usr/bin/pip3.7.5, /bin/ln
```

Replace `username` with the name of the non-root user who executes the installation script.

**NOTE**

Check if the last line in the `/etc/sudoers` file is `#includedir /etc/sudoers.d`. If no, add it manually.

3. Run the `:wq!` command to save the file.

4. Remove the write permission on the `/etc/sudoers` file:

```
chmod u-w /etc/sudoers
```

## Setting Up Environment

**NOTE**

This tool supports Python 3.7.0 to Python 3.7.9. This document uses Python 3.7.5 as an example. The environment variables and installation commands vary according to the actual Python version.

AMCT runs on Ubuntu 18.04 (x86_64) and EulerOS (AArch64). The following table lists the architecture mapping of Ubuntu 18.04 (x86_64) servers.

### Table 3-4 Ubuntu (x86_64) architecture mapping

<table>
<thead>
<tr>
<th>Item</th>
<th>Version</th>
<th>How to Obtain</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Ubuntu 18.04 (64-bit)</td>
<td>Click <a href="#">here</a> to download an Ubuntu release. The <strong>ubuntu-18.04-server-amd64.iso</strong> server install image is recommended.</td>
<td>-</td>
</tr>
<tr>
<td>Python</td>
<td>3.7.5</td>
<td>See <strong>3.11.5 Python 3.7.5 Installation on Ubuntu.</strong></td>
<td>Make sure that the server has internet access.</td>
</tr>
<tr>
<td>Caffe</td>
<td>Caffe-master branch</td>
<td>Currently, only the version whose commit ID is <strong>9b891540183d dc834a02b2bd 81b31afaee71b 2153</strong> is supported.</td>
<td>-</td>
</tr>
</tbody>
</table>

Prepare the Caffe environment by referring to the [Caffe tutorial](#).

You are advised to install the Caffe environment in source code mode. If the installation is performed in CLI mode and information similar to "'/usr/bin/python3.7: can't open file '/usr/lib/python3.7/py_compile.py': [Error 2] No such file or directory" is displayed, rectify the fault by referring to **3.10.1 Failed to Install the Caffe Environment in CLI Mode.**
<table>
<thead>
<tr>
<th>Item</th>
<th>Version</th>
<th>How to Obtain</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA Toolkit/ CUDA Driver</td>
<td>10.0</td>
<td>Obtain required packages for installation. For example, you can obtain the Toolkit package from the following URL, which contains the Driver package. <a href="https://developer.nvidia.com/cuda-toolkit-archive">https://developer.nvidia.com/cuda-toolkit-archive</a></td>
<td>To perform GPU quantization, the CUDA software must be installed.</td>
</tr>
<tr>
<td>NumPy</td>
<td>1.16.0+</td>
<td>See Installing Dependencies.</td>
<td>-</td>
</tr>
<tr>
<td>OpenCV-Python</td>
<td>4.1.0.25+</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Pillow</td>
<td>6.0.0+</td>
<td></td>
<td>Pillow 7.0.0 does not support the JPEG format.</td>
</tr>
<tr>
<td>wget</td>
<td>3.2+</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Protobuf</td>
<td>3.11.0+</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

### Checking Sources

During dependency installation, you need to make sure that the server of AMCT has Internet access. Run the following command as the root user to check source validity:

```bash
apt-get update
```

If an error is reported during the command execution, check whether the network connection is normal or replace the source in the `/etc/apt/sources.list` file with a valid one.

### Installing Dependencies

Use the AMCT installation user to install software. If the installation user is a non-root user, run the `su - username` command to switch to the non-root user and run the following commands.

### Table 3-5 Dependency list

<table>
<thead>
<tr>
<th>Required Component</th>
<th>Dependency</th>
<th>Version</th>
<th>Installation Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCT</td>
<td>Python</td>
<td>3.7.5</td>
<td>See 3.11.5 Python 3.7.5 Installation on Ubuntu.</td>
</tr>
<tr>
<td>Required Component</td>
<td>Dependency</td>
<td>Version</td>
<td>Installation Command</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td>---------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Protobuf</td>
<td></td>
<td>3.11.0</td>
<td>pip3.7.5 install protobuf==3.11.0 --user</td>
</tr>
<tr>
<td>Image classification network/MNIST</td>
<td>NumPy</td>
<td>1.16.0+</td>
<td>pip3.7.5 install numpy==1.16.0 --user</td>
</tr>
<tr>
<td>Image classification network/MNIST</td>
<td>OpenCV-Python</td>
<td>4.2.0.32</td>
<td>pip3.7.5 install opencv-python==4.2.0.32 --user</td>
</tr>
<tr>
<td>Image classification network</td>
<td>scikit-image</td>
<td>0.16.2</td>
<td>pip3.7.5 install scikit-image==0.16.2 --user</td>
</tr>
<tr>
<td>Image classification network</td>
<td>LMDB</td>
<td>0.98</td>
<td>pip3.7.5 install lmdb==0.98 --user</td>
</tr>
<tr>
<td>Detection network</td>
<td>2to3</td>
<td>-</td>
<td>sudo apt-get install -y 2to3</td>
</tr>
<tr>
<td>Detection network</td>
<td>Cython</td>
<td>0.29.15</td>
<td>pip3.7.5 install Cython==0.29.15 --user</td>
</tr>
<tr>
<td>Detection network</td>
<td>Matplotlib</td>
<td>3.2.0</td>
<td>pip3.7.5 install matplotlib==3.2.0 --user</td>
</tr>
<tr>
<td>Detection network</td>
<td>EasyDict</td>
<td>1.9</td>
<td>pip3.7.5 install easydict==1.9 --user</td>
</tr>
<tr>
<td>Detection network</td>
<td>PyYAML</td>
<td>5.3</td>
<td>pip3.7.5 install PyYAML==5.3 --user</td>
</tr>
<tr>
<td>Detection network</td>
<td>Pillow</td>
<td>6.0.0+</td>
<td>pip3.7.5 install pillow==6.0.0 --user</td>
</tr>
<tr>
<td>Detection network</td>
<td>pycocotools</td>
<td>2.0.2</td>
<td>pip3.7.5 install pycocotools==2.0.2 --user</td>
</tr>
<tr>
<td>MNIST</td>
<td>wget</td>
<td>3.2</td>
<td>pip3.7.5 install wget==3.2 --user</td>
</tr>
</tbody>
</table>

**Uploading the AMCT Package**

Upload the `Ascend-cann-amct_{software version}_ubuntu18.04-x86_64.tar.gz` package to any directory (for example, `$HOME/amct/`) on the Linux server as the AMCT installation user.

Decompress the AMCT package:

```
tar -zxvf Ascend-cann-amct-{software version}_ubuntu18.04-x86_64.tar.gz
```
## Find the following extracted packages.

### Table 3-6 Extracted parts of the AMCT package

<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
<tbody>
<tr>
<td>amct_caffe/</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
|                   | amct_caffe-{version}-py3-none-linux_{arch}.whl | Caffe AMCT package | - OS support: Ubuntu 18.04 (x86_64)  
- For details, see AMCT Instructions (Caffe).  
- Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
|                   | amct_caffe_sample.tar.gz | Caffe quantization sample package | |
|                   | caffe_patch.tar.gz | Caffe patch package | |
| amct_tensorflow/  |                   |             |                        |
|                   | amct_tensorflow-{version}-py3-none-linux_{arch}.whl | TensorFlow AMCT package | - OS support: Ubuntu 18.04 (x86_64)  
- amct_tensorflow and amct_tensorflow_ascend cannot exist at the same time.  
- For details, see AMCT Instructions (TensorFlow).  
- Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
|                   | amct_tensorflow_sample.tar.gz | TensorFlow quantization sample package | |
|                   | amct_tensorflow_ascend-{version}-py3-none-linux_{arch}.whl | TF Adapter AMCT package | - OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
- amct_tensorflow and amct_tensorflow_ascend cannot exist at the same time.  
- For details, see AMCT Instructions (TensorFlow, Ascend).  
- Inference on a quantized model needs to be |
<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
<tbody>
<tr>
<td>amct_tensorflow_ascend_sample.tar.gz</td>
<td>Package of quantization samples using TF Adapter</td>
<td>performed in the Ascend 310 inference environment installed with the Ascend AI Processor.</td>
<td></td>
</tr>
</tbody>
</table>
| amct_pytorch/ | PyTorch AMCT directory | | OS support: Ubuntu 18.04 (x86_64)  
For details, see AMCT Instructions (PyTorch).  
Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| amct_pytorch-{version}-py3-none-linux_{arch}.tar.gz | PyTorch AMCT source package | | |
| amct_pytorch_sample.tar.gz | PyTorch quantization sample package | | |
| amct_onnx/ | ONNX AMCT directory | | OS support: Ubuntu 18.04 (x86_64)  
For details, see AMCT Instructions (ONNX).  
Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| amct_onnx-{version}-py3-none-linux_{arch}.whl | ONNX AMCT package | | |
| amct_onnx_op.tar.gz | ONNX Runtime AMCT custom OPP | | |
| amct_onnx_sample.tar.gz | ONNX quantization sample package | | |
| amct_mindspore/ | MindSpore AMCT directory | | OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
For details, see AMCT Instructions (MindSpore).  
Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| amct_mindspore-{version}-py3-none-linux_{arch}.whl | MindSpore AMCT package | | |
| amct_mindspore_sample.tar.gz | MindSpore quantization sample package | | |
| amct_ascendcl/ | AscendCL API AMCT directory | | OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
For details, see AMCT Instructions (AscendCL). |
| Ascend-amct_acl-{software version}-{os}_{arch}.run | AscendCL API AMCT package | | |
3.2.3 Installation

Step 1  In the directory where the AMCT package is located, run the following command:

```
pip3.7.5 install amct_caffe-{version}-py3-none-linux_{arch}.whl --user
```

Replace `{version}` with the actual AMCT version number, and `{arch}` with the actual architecture of the installation server. If AMCT installation is performed by the root user and the `--target` option is included, ensure that the path specified by `--target` is the path of the current user.

Step 2  Check the installation. If a message similar to the following is displayed, the installation is successful.

```
Successfully installed amct-caffe-{version}
```

Find the installed AMCT in the `python3.7.5` directory (for example, `$HOME/.local/lib/python3.7.5/site-packages`).

```
  drwxr-xr-x  5 amct amct   4096 Mar 17 11:50 amct_caffe
  drwxr-xr-x  2 amct amct   4096 Mar 17 11:50 amct_caffe-{version}.dist-info/
```

`amct_caffe` indicates the AMCT installation path.

---End

3.2.4 Post-installation Actions

3.2.4.1 Installing Patches

After the AMCT installation is complete, you need to obtain and install the Caffe patch package `caffe_patch.tar.gz` for the following purposes:

- If a `custom.proto` file exists on the AMCT server, merge the `custom.proto` file with the `.proto` file in the AMCT package. This package provides the `caffe.proto` file based on Caffe 1.0, the AMCT custom layers, and the `amct_custom.proto` file of Caffe-master updated layers compared to Caffe 1.0. For details about `.proto` file merging, see 3.11.1 Proto Merging Principles.

- Copy the new source code and dynamic library files to the Caffe-master project directory in the Caffe environment.
- Install the patches for some files in the Caffe-master project directory in the Caffe environment to automatically modify the files.

**Prerequisites**

You have prepared the `custom.proto` file and uploaded it to any directory on the AMCT server. An example is as follows.

```protobuf
message LayerParameter {
  optional ReLU6Parameter relu6_param = 2060;
  optional ROIPoolingParameter roi_pooling_param = 8266711;
}

message ReLU6Parameter {
  optional float negative_slope = 1 [default = 0];
}

message ROIPoolingParameter {
  // Pad, kernel size, and stride are all given as a single value for equal
  // dimensions in height and width or as Y, X pairs.
  optional uint32 pooled_h = 1 [default = 0]; // The pooled output height
  optional uint32 pooled_w = 2 [default = 0]; // The pooled output width
  // Multiplicative spatial scale factor to translate ROI coords from their
  // input scale to the scale used when pooling
  optional float spatial_scale = 3 [default = 1];
}
```

The `custom.proto` file consists of two parts:

1. Register a custom layer with **LayerParameter**.

   ```protobuf
   message LayerParameter {
     # user definition fields, each field takes one line.
     optional FieldType0 field_name0 = field_num0;
     optional FieldType1 field_name1 = field_num1;
   }
   ```

   This field is used to declare the custom layer in **LayerParameter**. The custom layer needs to be added to **LayerParameter** so that it can be written to and read from the layer in the Caffe framework. This declaration is divided into four parts:
   - **optional**: indicates that the definition is optional in **LayerParameter**. Must be **optional**.
   - **FieldType**: declares the custom type of the current field. The corresponding message definition is required.
   - **field_name**: indicates the ID of the current declaration, which must be unique. If a conflict occurs, you need to change the ID. The ID will be used to access the corresponding content.
   - **field_num**: indicates the current declaration index number, which must be unique. If a conflict occurs, you need to change the index number. Pass a number less than 5000 that does not conflict with that in the ATC `caffe.proto` file. In the binary `.caffemodel` file, the corresponding field needs to be parsed based on the index number.

   The following is an example.

   ```protobuf
   message LayerParameter {
     optional ReLU6Parameter relu6_param = 2060;
     optional ROIPoolingParameter roi_pooling_param = 8266711;
   }
   ```
NOTE

- Keep the sequence number a custom layer in custom.proto within 5000 and avoid conflict with the built-in layers in the ATC caffe.proto file.
- The layers in amct_custom.proto are numbered starting at 200000.
- The sequence numbers of the ATC custom layers in caffe.proto are within the range [5000, 200000].

2. Define the parameters of a custom layer.

```protobuf
class ReLU6Parameter {
  optional float negative_slope = 1 [default = 0];
}
```

For details, see Google Protobuf.

Ensure that the value of this field does not conflict with amct_custom.proto at the custom layer of AMCT. If they conflict, an error message will be displayed during .proto merging. You can modify the value based on the error message. If it conflicts with the built-in caffe.proto of ATC, the message definition applies.

Currently, the messages customized by AMCT include QuantParameter, DeQuantParameter, IFMRRParameter, LSTMQuantParameter, SearchNParameter, RetrainDataQuantParameter, RetrainWeightQuantParameter, SingleLayerRecord, and ScaleOffsetRecord. The names of the custom layers cannot be the same as those of the preceding messages.

Procedure

You can run the auto installation script install.py in caffe_patch. If the script is run successfully, the patches in caffe_patch are automatically installed in the Caffe-master project directory of the Caffe environment, and the .proto files are merged. The function of replacing the source code and dynamic library files is newly supported. After the installation or manual modification is complete, you need to recompile the Caffe environment. The procedure is as follows:

**Step 1** Decompress the Caffe patch package.

Run the following command as the AMCT installation user to decompress the caffe_patch.tar.gz software package:

```
tar -zxvf caffe_patch.tar.gz
```

Find the following extracted directories and files:
- **caffe_patch/include**: directory of header files and common functions of the custom layer.
- **caffe_patch/install.py**: script for merging .proto files, installing patches, and executing source code and dynamic library files in the Caffe environment.
- **caffe_patch/merge_proto**: directory of merged .proto file.
- **caffe_patch/patch**: directory of patches.
- **caffe_patch/quant_lib**: directory of core dynamic libraries libquant.so and libquant_gpu.so of the quantization algorithm.
- **caffe_patch/src**: directory of implementation source code files and common functions of the custom layer.
Step 2  Switch to the directory where the `caffe_patch/install.py` script is stored and run the following command:

```
python3.7.5 install.py --caffe_dir CAFFE_DIR [--custom_proto CUSTOMPROTO_FILE]
```

The command-line options are described as follows.

**Table 3-7 Command-line options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--h</td>
<td>(Optional) Displays help information.</td>
</tr>
<tr>
<td>--caffe_dir CAFFE_DIR</td>
<td>(Required) Sets the directory of the Caffe source code, which can be relative or absolute.</td>
</tr>
<tr>
<td>--custom_proto CUSTOMPROTO_FILE</td>
<td>(Optional) Sets the directory of the <code>custom.proto</code> file, which can be relative or absolute.</td>
</tr>
</tbody>
</table>

An example is as follows:

```
python3.7.5 install.py --caffe_dir caffe-master --custom_proto custom.proto
```

If messages similar to the following are displayed, the execution is successful:

```
# Copy the new source code and dynamic library files to the Caffe-master project directory in the Caffe environment.
[INFO]Begin to copy source files, header files and quant_lib to 'HOME/AMCT/AMCT_CAFFE/caffe-master'
[INFO]Finish copy source files, header files and quant_lib to 'HOME/AMCT/AMCT_CAFFE/caffe-master'
[INFO]Begin to install patch.
[INFO]Install patch 'lstm_calibration_layer.cpp.patch' successfully.
[INFO]Install patch 'lstm_quant_layer.hpp.patch' successfully.
[INFO]Install patch 'lstm_quant_layer.cpp.patch' successfully.
[INFO]Install patch 'lstm_quant_layer.hpp.patch' successfully.
[INFO]Finish install patch.
# Merge the .proto files.
[INFO]Merge and replace "caffe.proto" success.
# Modify the Makefile.
```

The `install.py` script allows repeated installation of patches.

- If a patch fails to be installed, restore the `caffe-master/src/caffe/layers/lstm_layer.cpp` and `caffe-master/include/caffe/layers/lstm_layer.hpp` files in the Caffe project to the native files of Caffe-master.
- If an error message is displayed during .proto merging, rectify the fault by referring to 3.10.3 Proto Merging Error.
- If Makefile fails to be modified, modify it as prompted. If the script is executed successfully, Makefile will not be modified repeatedly when the script is executed again.

Step 3  (Optional) This step takes effect only for the object detection network. Skip this step if you are not using an object detection sample.
Modify the `caffe-master/src/caffe/proto/caffe.proto` file to add a customized layer.

1. Add the following information to the end of `message LayerParameter`:
   ```
   optional ROIPoolingParameter roi_pooling_param = 8266711;
   ``
2. Add the following information to the end of the file:
   ```
   // Message that stores parameters used by ROIPoolingLayer
   message ROIPoolingParameter {
     // Pad, kernel size, and stride are all given as a single value for equal
     // dimensions in height and width or as Y, X pairs.
     optional uint32 pooled_h = 1 [default = 0]; // The pooled output height
     optional uint32 pooled_w = 2 [default = 0]; // The pooled output width
     // Multiplicative spatial scale factor to translate ROI coords from their
     // input scale to the scale used when pooling
     optional float spatial_scale = 3 [default = 1];
   }
   ``
3. Switch to the `caffe-master` directory and modify the `caffe-master/Makefile.config` file.
   ```
   # Uncomment to support layers written in Python (will link against Python libs)
   WITH_PYTHON.Layer := 1
   ```

**Step 4** The C++11 standard code is supported.

The new AMCT operators require the C++11 support. Therefore, add the `-std=C++11` compile option to `caffe-master/Makefile`:

```
# Complete build flags.
COMMON_FLAGS += $(foreach includedir,$(INCLUDE_DIRS),-I$(includedir)) --std=c++11
CXXFLAGS += -pthread -fPIC $(COMMON_FLAGS) $(WARNINGS)
NVCCFLAGS += -ccbin=$(CXX) -Xcompiler -fPIC $(COMMON_FLAGS)
```

**Step 5** Go back to the `caffe-master` directory and run the following commands to recompile the Caffe and PyCaffe environments:

```make
# If the Caffe project has been built before the patches are installed in the user environment, run the make clean command and then the build command after the patches are installed.
make clean
make all -j && make pycaffe -j
```

After `caffe.proto` is modified, it needs to be recompiled into `caffe_pb2.py`. Because the Caffe model needs to be parsed, a custom layer may be added when the Caffe model is used. In this case, you need to modify the `caffe.proto` file. After the modification: You need to provide `caffe_pb2.py` built from the modified `caffe.proto` file for AMCT.

**NOTE**

If you use the protoc mode to rebuild `caffe.proto`, for example, `protoc --python_out=./ caffe.proto`, you need to change the path of `caffe.proto` in `PYTHONPATH`. Replace `$path` with the actual path of `caffe.proto`.

```export PYTHONPATH=$(PYTHONPATH):$path
```

```----End```

**3.2.4.2 Setting Environment Variables**

Set the log print level. Logs include the logs printed to the screen and the logs saved in the `amct_log/amct_caffe.log` file. The environment variables are optional. If they are not set, the default log level INFO is used.
Variables

The log level is set by the following variables:

- **AMCT_LOG_FILE_LEVEL**: specifies the level of messages in the `amct_caffe.log` file and the level of messages in the log file generated of the corresponding quantization layer when the model for accuracy simulation is generated.

- **AMCT_LOG_LEVEL**: specifies the level of log messages printed to the screen.

*Table 3-8* lists the valid values and their meanings.

<table>
<thead>
<tr>
<th>Logging Level</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG</td>
<td>Outputs DEBUG, INFO, WARNING, and ERROR logs.</td>
<td>Detailed process messages, including the quantization layer and corresponding processing phase (fusion, parameter quantization, or activation quantization)</td>
</tr>
<tr>
<td>INFO</td>
<td>Outputs INFO, WARNING, and ERROR logs. The default value is INFO.</td>
<td>Brief quantization processing messages, including the quantization phase.</td>
</tr>
<tr>
<td>WARNING</td>
<td>Outputs WARNING and ERROR logs.</td>
<td>Warning messages during quantization.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Outputs ERROR logs.</td>
<td>Error messages during quantization.</td>
</tr>
</tbody>
</table>

The logging level is case insensitive. That is, *Info*, *info*, and *INFO* are all valid values.

**Example**

The following commands are only examples. You can set the level as required.

- Set the quantization log level of `amct_caffe.log` to *INFO*.
  ```bash
  export AMCT_LOG_FILE_LEVEL=INFO
  ```

- Set the level of the information displayed on the screen to *INFO*.
  ```bash
  export AMCT_LOG_LEVEL=INFO
  ```

### 3.3 Post-training Quantization

#### 3.3.1 Sample Code

This section explains the post-training quantization template code line by line to facilitate your understanding of the AMCT workflow. With only small tweaks, you can adapt the template code to more network models.
Prerequisites

In the directory of the quantization sample package **amct_caffe_sample.tar.gz**, decompress the package:

```
tar -zxvf amct_caffe_sample.tar.gz
cd sample
```

Find the following extracted files and directories:

- **amct_caffe_calibration_template.py**: post-training quantization code template.
- **resnet50/**: quantization directory of the image classification network ResNet-50. For details, see 3.3.2.1 Quantizing an Image Classification Network or 3.4 Quantization Aware Training.
- **faster rcnn/**: quantization directory of the object detection network Faster R-CNN. For details, see 3.3.2.2 Quantizing an Object Detection Network.
- **mnist/**: quantization directory of the MNIST network. For details, see 3.3.2.4 Quantizing the MNIST Network.
- **tensor decompose/**: tensor decomposition directory. For details, see 3.6 Tensor Decomposition.
- **mobilenetV2/**: directory of accuracy-oriented automatic quantization rollback. For details, see 3.5 Accuracy-oriented Automatic Quantization Rollback.

AMCT Workflow

1. Set the compute mode and device.
   
   The AMCT uses the **amct.set_gpu_mode()** and **amct.set_cpu_mode()** APIs, which are closely related to the Caffe framework. In GPU mode, the Caffe APIs **caffe.set_mode_gpu()** and **caffe.set_device(args.gpu_id)** are used. Therefore, you need to configure the Caffe compute mode and device before configuring the AMCT mode. In addition, because the compute device has been specified here, you do not need to configure the compute device in the model inference function. A code example is as follows.

   ```python
   if args.gpu_id is not None and not args.cpu_mode:
       caffe.set_mode_gpu()
       caffe.set_device(args.gpu_id)
       amct.set_gpu_mode()
   else:
       caffe.set_mode_cpu()
   ```

2. Run the original Caffe model for inference to validate the inference script and environment.

   ```python
   # Run original model without quantize test
   if args.pre_test:
       run_caffe_model(args.model_file, args.weights_file, args.iterations)
       print('[INFO]Run %s without quantize success!' %(args.model_name))
       return
   ```

3. Parse the user model and generate a full quantization configuration file.
   
   a. If the configuration file is generated by tweaking the simplified configuration file, set the **config_definition** parameter. The rest parameters are invalid and do not need to be set.
   
   b. You can pass the **skip_layers**, **batch_num**, and **activation_offset** arguments to the API call to generate a quantization configuration file. A code example is as follows.
# Generate quantize configurations
config_json_file = 'tmp/config.json'
batch_num = 2
if args.cfg_define is not None:
    amct.create_quant_config(config_json_file,
        args.model_file,
        args.weights_file,
        config_defination=args.cfg_define)
else:
    skip_layers = []
    amct.create_quant_config(config_json_file,
        args.model_file,
        args.weights_file,
        skip_layers,
        batch_num)

# Phase0: Init amct task
scale_offset_record_file = 'tmp/scale_offset_record.txt'
graph = amct.init(config_json_file,
    args.model_file,
    args.weights_file,
    scale_offset_record_file)

# Phase1: Do conv+bn+scale fusion, weights calibration and fake quantize, insert data-quantize layer
modified_model_file = 'tmp/modified_model.prototxt'
modified_weights_file = 'tmp/modified_model.caffemodel'
amct.quantize_model(graph, modified_model_file, modified_weights_file)

# Phase2: run caffe model to do activation calibration
run_caffe_model(modified_model_file, modified_weights_file, batch_num)

# Phase3: save final model, one for caffe do fake quant test, one deploy model for ATC
result_path = 'results/%s' %(args.model_name)
amct.save_model(graph, 'Both', result_path)

e. (Optional) Perform inference using the accuracy simulation model (fake_quant) to test the model accuracy.
   # Phase4: if need test quantized model, uncomment to do final fake quant
   # model test.
   fake_quant_model = 'results/%s_fake_quant_model.prototxt'.format(args.model_name)
fake_quant_weights = 'results/%s_fake_quant_weights.caffemodel'.format(args.model_name)
run_caffe_model(fake_quant_model, fake_quant_weights, args.iterations)

---

**Manual Tweaking**

1. Modify the arguments.

Pass the necessary argument for executing the AMCT. This step is optional. You can implement this in your own ways. The following is a code example.

class Args(object):
    """struct for Args"""
    def __init__(self):
self.model_name = "" # Caffe model name as prefix to save model
self.model_file = "" # user caffe model txt define file
self.weights_file = "" # user caffe model binary weights file
self.cpu = True # If True, force to CPU mode, else set to False
self.gpu_id = 0 # Set the gpu id to use
self.pre_test = False # Set true to run original model test, set False to run quantize with amct_caffe tool
self.iterations = 5 # Iteration to run caffe model
self.cfg_define = None # If None use

args = Args()

# User set basic info to use amct_caffe tool

# e.g.
args.model_name = 'ResNet50'
args.model_file = 'pre_model/ResNet-50-deploy.prototxt'
args.weights_file = 'pre_model/ResNet-50-model.caffemodel'
args.cpu = True
args.gpu_id = None
args.pre_test = False
args.iterations = 5
args.cfg_define = None

2. Modify the code lines for performing Caffe model inference.

The following is a code example.

def run_caffe_model(model_file, weights_file, iterations):
    net = caffe.Net(model_file, weights_file, caffe.TEST)

    # User modified to execute caffe model forward
    # e.g.
    # for iter_num in range(iterations):
    #     data = get_data()
    #     forward_kwargs = {'data': data}
    #     blobs_out = net.forward(**forward_kwargs)
    #     # if have label and need check network forward result
    #     post_process(blobs_out)
    # return

The code is described as follows. Implement model inference based on the specific service network.

a. Pass the model file to create a Caffe network (set phase to caffe.TEST for inference).
    net = caffe.Net(model_file, weights_file, caffe.TEST)

b. Set iterations to the number of inference iterations.

c. Obtain the network data required every iteration. Preprocess the data based on the service network. For example, for ResNet-50, you need to convert YUV images to the RGB format, resize to 224 x 224, and subtract the mean values from each channel. Then, construct input blobs in key(blob name):value(NumPy array) dictionary format.

data = get_data()
forward_kwargs = {'data': data}

blobs_out = net.forward(**forward_kwargs)

# if have label and need check network forward result
post_process(blobs_out)

return
classification or object detection result. This step is not required by AMCT. You only need to perform network inference to obtain all intermediate-layer data of the network. You can determine whether to perform postprocessing on the network inference result.

```
post_process(blobs_out)
```

### 3.3.2 Uniform Quantization

This section describes how to perform uniform quantization on a Caffe network by using the quantization script.

#### 3.3.2.1 Quantizing an Image Classification Network

##### 3.3.2.1.1 Quantization Preparations

**Model**

Upload the Caffe model to quantize and its weight file to any directory on the Linux server as the AMCT installation user. This section uses the image classification network ResNet-50 available in the sample package as an example. Download the model file in advance.

Go to the `sample/resnet50` directory and run the following command to download the `ResNet-50-deploy.prototxt` model file:

```
python3.7.5 download_prototxt.py --caffe_dir CAFFE_DIR --close_certificate_verify
```

For details about the available command-line options, see [Table 3-9](#).

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--h</code></td>
<td>(Optional) Displays help information.</td>
</tr>
<tr>
<td><code>--caffe_dir CAFFE_DIR</code></td>
<td>(Required) Sets the directory of the Caffe source code, which can be relative or absolute.</td>
</tr>
<tr>
<td><code>--close_certificate_verify</code></td>
<td>(Optional) Disables certificate validation to ensure successful download.</td>
</tr>
<tr>
<td></td>
<td>If a message is displayed indicating that authentication fails during model download, include this option to your download command and try again.</td>
</tr>
</tbody>
</table>

An example is as follows:

```
python3.7.5 download_prototxt.py --caffe_dir caffe-master --close_certificate_verify
```

If messages similar to the following are displayed, the model file is successfully downloaded:

```
```
You can view the downloaded model in the `sample/resnet50/pre_model` directory as prompted. `ResNet-50_retrain.prototxt` is the model file used in the quantization aware training scenario. For details, see 3.4 Quantization Aware Training.

Image Dataset

After the model is quantized using the AMCT, perform inference with the model to test the model accuracy. Use the dataset that matches the model.

Upload the dataset matching the model to any directory on the Linux server as the AMCT installation user. This section uses the images dataset corresponding to the ResNet-50 network in the sample package as an example.

Calibration Dataset

The calibration dataset is used to generate the quantization factors to guarantee the accuracy.

The process of calculating the quantization factors is referred to as calibration. Perform inference using the quantized network with one or more batches of a subset of images from the validation dataset to complete calibration. To ensure the quantization accuracy, the source of the calibration dataset must be the same as that of the validation dataset.

Upload the calibration dataset file to any directory on the Linux server as the AMCT installation user.

3.3.2.1.2 Quantization Example

There are two quantization modes. One is to use the `ResNet50_sample.py` quantization script that allows much flexibility. The other is to use the encapsulated script `run_resnet50_with_arq.sh` where only a few parameters need to be configured. Select the script that best suits your requirement.

1. Run the quantization script.
   - `ResNet50_sample.py` quantization script
     i. Precheck the original network to test if it can run properly in the Caffe environment.

     This step is added to identify risks in advance such as dataset and model mismatch and model execution failures in the Caffe environment.

     Run the following command in the directory of the quantization script to precheck the ResNet-50 network:

     ```
     ```

     Table 3-10 describes the command-line options.

### Table 3-10 Command-line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--h</td>
<td>(Optional) Displays help information.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--model_file MODEL_FILE</td>
<td>(Required) Sets the directory of the Caffe model file (.prototxt).</td>
</tr>
<tr>
<td>--weights_file WEIGHTS_FILE</td>
<td>(Required) Sets the directory of the Caffe weight file (.caffemodel).</td>
</tr>
<tr>
<td>--gpu GPU_ID</td>
<td>(Optional) Sets the ID of the compute GPU device.</td>
</tr>
<tr>
<td>NOTE</td>
<td>In GPU inference mode, compile the Caffe environment of the GPU version before running the quantization script.</td>
</tr>
<tr>
<td>--cpu</td>
<td>(Optional) Enables the CPU inference mode.</td>
</tr>
<tr>
<td>--iterations ITERATIONS</td>
<td>(Optional) Sets the batch count for inference.</td>
</tr>
<tr>
<td>--caffe_dir CAFFE_DIR</td>
<td>(Required) Sets the directory of the Caffe source code, which can be relative or absolute.</td>
</tr>
<tr>
<td>--pre_test</td>
<td>(Optional) Prechecks the original model and provides the inference result if it can run properly in the Caffe environment.</td>
</tr>
<tr>
<td>--cfg_define CFG_DEFINE</td>
<td>(Optional) Sets the configuration file path. Required for non-uniform quantization.</td>
</tr>
<tr>
<td>--benchmark</td>
<td>(Optional) Uses the benchmark ImageNet dataset for quantization. Required for model accuracy analysis.</td>
</tr>
<tr>
<td>--dataset DATASET</td>
<td>(Optional) Sets the directory of the ImageNet dataset in LMDB format. Required for model accuracy analysis.</td>
</tr>
</tbody>
</table>

An example is as follows.

```python
```

If a message similar to the following is displayed, the original model runs properly in the Caffe environment.

```
[AMCT][INFO]Run ResNet-50 without quantize success!
```

ii. Run the quantization script to quantize the original network.

```python
```
If messages similar to the following are displayed, the model is successfully quantized. (The top 1 and top 5 inference accuracy results are examples only.)

```
******final top1:0.86875
******final top5:0.95     //Top 1 and top 5 inference accuracy results of the quantized fake_quant model in the Caffe environment.
[AMCT][INFO]Run ResNet-50 with quantize success!
```

**NOTE**

When quantizing an original third-party network on GPU, if a message is displayed indicating that the GPU resources are insufficient, as shown in the following figure, take the following steps to fix the problem:

1. Use a GPU with larger Video RAM.
2. Check if there are other processes sharing the GPU resources. If yes, wait until the GPU resources are idle.
3. If the memory is sufficient, switch to the CPU mode.

- **run_resnet50_with_arq.sh** encapsulated quantization script
  
  The **run_resnet50_with_arq.sh** script in the `sample/resnet50/scripts` directory has encapsulated the `ResNet50_sample.py` quantization script and minimized the parameters to be configured.

  Run the following command in the `sample/resnet50` directory:

  ```bash
  bash scripts/run_resnet50_with_arq.sh -c your_caffe_dir -g gpu_id
  ```

  The command-line options are described as follows.

  **Table 3-11 Command-line options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>(Required) Caffe-master directory.</td>
</tr>
<tr>
<td>-g</td>
<td>(Optional) GPU device ID. If it is not specified, quantization runs on the CPU by default.</td>
</tr>
</tbody>
</table>

An example is as follows.

```
bash scripts/run_resnet50_with_arq.sh -c caffe-master -g 0
```

If messages similar to the following are displayed, the model is successfully quantized. (The top 1 and top 5 inference accuracy results are examples only.)

```
******final top1:0.86875
******final top5:0.95     //Top 1 and top 5 inference accuracy results of the quantized fake_quant model in the Caffe environment.
[AMCT][INFO]Run ResNet-50 with quantize success!
```

2. **View the quantization result.**

After the quantization is complete, the inference result using the accuracy simulation model obtained after quantization is displayed. Find the following files generated in the directory of the quantization script:

- **amct_log/amct_caffe.log**: AMCT log file.
- **tmp**: quantization temporary folder, containing:
- **config.json**: configuration file that describes how to quantize each layer in the model. If a quantization configuration file already exists in the directory of the quantization script, the existing quantization configuration file is overwritten by a new one with the same name in a call to `create_quant_config`. If not that case, a new quantization configuration file is created. If the accuracy of model inference drops significantly after quantization, you can modify the `config.json` file by referring to 3.3.3 Quantization Configuration.

- **modified_model.prototxt** and **modified_model.caffemodel**: intermediate model files.

- **scale_offset_record.txt**: file that records quantization factors. For details about the prototype definition of the file, see 3.11.4 Quantization Factor Record File.

  - **results/calibration_results**: quantization result folder, containing the quantized model file as well as its weight file and quantization information file `ResNet50_quant.json` (named after the quantized model).

- **ResNet50_deploy_model.prototxt**: quantized model file to be deployed on the Ascend AI Processor.

- **ResNet50_deploy_weights.caffemodel**: weight file of the quantized model to be deployed on the Ascend AI Processor.

- **ResNet50_fake_quant_model.prototxt**: quantized model file for accuracy simulation in the Caffe environment.

- **ResNet50_fake_quant_weights.caffemodel**: weight file of the quantized model file for accuracy simulation in the Caffe environment.

- **ResNet50_quant.json**: quantization information file (named after the quantized model). This file gives the node mapping between the quantized model and the original model and is used for accuracy comparison between the quantized model and the original model.

  When a model is re-quantized, the existing result files will be overwritten.

3. (Optional) Convert the quantized deployable model into an offline model adapted to the Ascend AI Processor by referring to ATC Instructions.

### 3.3.2.1.3 Model Accuracy Analysis

Inference and quantization calibration in 3.3.2.1.2 Quantization Example are performed based on the built-in image dataset. Therefore, the quantization result is used only to verify whether the model is successfully quantized and cannot be used to validate the model accuracy after quantization. This section describes how to compare the model accuracy before and after quantization based on the benchmark ImageNet dataset.

Download the benchmark ImageNet dataset and convert it to the LMDB format in advance.
Preparations

Download the ImageNet dataset and convert it to LMDB format by referring to the `caffe-master/examples/imagenet/readme.md` file of the Caffe project.

Procedure

1. Test the accuracy of the original model (before quantization).
   
   Run the following command:
   ```
   ```
   
   For details about the available command-line options, see Table 3-10. If messages similar to the following are displayed, the execution is successful:
   ```
   ******final top1:0.725
   ******final top5:0.91875
   [AMCT][INFO]Run ResNet-50 without quantize success!
   ```

2. Test the accuracy of the quantized model.
   ```
   ```
   If messages similar to the following are displayed, the model is successfully quantized. (The top 1 and top 5 inference accuracy results are examples only.)
   ```
   ******final top1:0.7125
   ******final top5:0.925
   [AMCT][INFO]Run ResNet-50 with quantize success!
   ```

3. Compare the model accuracy before and after quantization based on the tested top 1 and top 5 accuracy results.

3.3.2.2 Quantizing an Object Detection Network

3.3.2.2.1 Quantization Preparations

Model

For details, see Model.

The Faster R-CNN network is automatically downloaded to the local host during Environment Initialization. This document uses the downloaded Faster R-CNN as an example. You can choose to prepare a network yourself.

Image Dataset

For details, see Image Dataset.

During Environment Initialization, the preset image dataset of the Faster R-CNN network is also downloaded to the local host.

Calibration Dataset

For details, see Calibration Dataset.
Environment Initialization

Initialize the environment to obtain the network source code, model files, weight files, and datasets.

1. Obtain necessary files of the object detection network.
   - If the server installed with the AMCT can access to the Internet and GitHub:
     Run the following script in `sample/faster_rcnn` in the network quantization package to initialize the environment:
     ```bash
     ```
     Table 3-12 describes the arguments in the preceding command. The following is an example of the command. Replace the arguments as required.
     ```bash
     bash init_env.sh CPU */caffe-master/ python3.7 /usr/include/python3.7m
     ```
   - If the user environment has no Internet access:
     i. On a server that has Internet access, download the packages from the given links and upload the packages to the `sample/faster_rcnn` directory:
        1) `faster_rcnn` script: available at [https://github.com/rbgirshick/caffe-fast-rcnn/archive/0dcd397b29507b8314e252e850518c5695efbb83.zip](https://github.com/rbgirshick/caffe-fast-rcnn/archive/0dcd397b29507b8314e252e850518c5695efbb83.zip) Rename the downloaded zip package `faster_rcnn_caffe_master.zip`. The faster_rcnn Caffe project is generated after the environment is initialized.
        2) `faster_rcnn caffe_master` project: available at [https://github.com/rbgirshick/py-faster-rcnn/archive/master.zip](https://github.com/rbgirshick/py-faster-rcnn/archive/master.zip). Rename the downloaded zip package `py-faster-rcnn-master.zip` and execute it to generate the faster_rcnn project package.
        3) `vgg16_faster_rcnn` pre-trained model file: available at [https://dl.dropboxusercontent.com/s/o6ii098bu51d139/faster_rcnn_models.tgz](https://dl.dropboxusercontent.com/s/o6ii098bu51d139/faster_rcnn_models.tgz). The vgg-16 faster_rcnn pre-trained model is generated after the environment is initialized.

     This dataset is used only for model accuracy test. For details, see 3.3.2.2.3 Model Accuracy Analysis.
     ii. Go to the `sample/faster_rcnn` directory and run the initialization script. The following is an example:
     ```bash
     bash init_env.sh CPU */caffe-master/ python3.7 /usr/include/python3.7m
     ```
     The initialization script first checks whether a downloaded package exists in the current directory. If the package exists, the initialization script will not attempt to download the package from online again. Instead, the existing package is used to generate files in the corresponding directory.
Table 3-12 Environment initialization script arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[h]</td>
<td>(Optional) Displays help information.</td>
</tr>
<tr>
<td>arg[1]</td>
<td>(Required) Sets the run mode, which can be CPU or GPU. <strong>NOTE</strong>&lt;br&gt;• In CPU mode, only the [--cpu] argument can be used in the quantization command.&lt;br&gt;• In GPU mode, the [--gpu GPU_ID] or [--cpu] argument can be used in the quantization command.&lt;br&gt;Select the argument for environment initialization as required.</td>
</tr>
<tr>
<td>arg[2]</td>
<td>(Required) Sets the Caffe-master absolute path.</td>
</tr>
<tr>
<td>arg[3]</td>
<td>(Optional) Sets the Python 3 version. The default value is python3. If there are multiple versions, you can use this argument to specify the Python 3 version, for example, python3.7.</td>
</tr>
<tr>
<td>arg[4]</td>
<td>(Optional) Sets the Python3m path, which must match the Python 3 version. The default value is /usr/include/python3.7m.</td>
</tr>
<tr>
<td>arg[5]</td>
<td>(Optional) To perform a model test, set this argument to with_benchmark. For details, see 3.3.2.2.3 Model Accuracy Analysis.</td>
</tr>
</tbody>
</table>

2. After the environment is initialized, the following directories are generated:
   a. caffe_master_patch: Caffe source code patch folder. You need to manually copy the following files to the Caffe-master project.
      - include/caffe/fast_rcnn_layers.hpp: header file of the custom layer definition.
      - src/caffe: folder of the implementation source files of the custom layer.
   
   Run the following command:
   ```bash
cp -r $HOME/amct/amct_caffe/sample/faster_rcnn/caffe_master_patch/* caffe-master/
```

   b. The following folders are added to the amct_caffe_faster_rcnn_sample directory:
      - datasets: folder of datasets used by Faster R-CNN.
      - pre_model: folder of the Faster R-CNN model file (faster_rcnn_test.pt) and weight file (VGG16_fasterrcnn_final.caffemodel).
      - python_tools: folder of Faster R-CNN source code.

3. Go back to the caffe-master directory and run the following command to recompile the Caffe environment:
   ```bash
   make clean
   make all & & make pycaffe
   ```
3.3.2.2 Quantization Example

1. Precheck the original network to test if it can run properly in the Caffe environment.

   This step is added to identify risks in advance such as dataset and model mismatch and model execution failures in the Caffe environment.

   Switch to the sample/faster_rcnn/amct_caffe_faster_rcnn_sample directory and run the following command to detect the faster_rcnn network:

   ```
   ```

   Table 3-10 describes the available command-line options.

   An example is as follows:

   ```
   python3.7.5 faster_rcnn_sample.py --model_file pre_model/faster_rcnn_test.pt --weights_file pre_model/VGG16_faster_rcnn_final.caffemodel --gpu 0 --pre_test
   ```

   The number of detection result files is displayed based on the number of detected objects in the dataset in amct_caffe_faster_rcnn_sample/datasets. When you close the detection result file, if messages similar to the following are displayed on the server where AMCT is located, the original model runs properly in the Caffe environment.

   ```
   [AMCT][INFO] Run faster_rcnn without quantize success!
   ```

   The precheck result file is stored in amct_caffe_faster_rcnn_sample/pre_detect_results/.

2. Run the quantization script.

   ```
   python3.7.5 faster_rcnn_sample.py --model_file pre_model/faster_rcnn_test.pt --weights_file pre_model/VGG16_faster_rcnn_final.caffemodel --gpu 0
   ```

   The number of detection result files is displayed based on the number of detected objects in the dataset in amct_caffe_faster_rcnn_sample/datasets. You can compare the bounding box position on the image with the inference result of the original model after the [--pre_test] option is used.

   Close all detection result files. A quantization success message is displayed on the AMCT server:

   ```
   [AMCT][INFO] Run faster_rcnn with quantize success!
   ```

   The postcheck result file is stored in amct_caffe_faster_rcnn_sample/quant_detect_results/.

3. View the quantization result.

   After the quantization is complete, the inference result using the accuracy simulation model obtained after quantization is displayed. Find the following files generated in the directory of the quantization script:

   - **config.json**: configuration file that describes how to quantize each layer in the model. If a quantization configuration file already exists in the directory of the quantization script, the existing quantization configuration file is overwritten by a new one with the same name in a call to `create_quant_config`. If not that case, a new quantization configuration file is created.

     If the accuracy of model inference drops significantly after quantization, you can modify the **config.json** file by referring to 3.3.3 Quantization Configuration.

   - **amct_log/amct_caffe.log**: AMCT log file.
- **pre_detect_results**: precheck result folder.
- **quant_detect_results**: postcheck result folder.
- **tmp**: quantization temporary folder, including the temporary model files `modified_model.prototxt` and `modified_model.caffemodel`, and the quantization factor file (`scale_offset_record/record.txt`). See 3.11.4 **Quantization Factor Record File** for the prototype definition of this file.
- **results**: quantization result folder, containing the quantized model file and its weight file as well as the quantization information file.
  - `faster_rcnn_deploy_model.prototxt`: quantized model file to be deployed on the Ascend AI Processor.
  - `faster_rcnn_deploy_weights.caffemodel`: weight file of the quantized model to be deployed on the Ascend AI Processor.
  - `faster_rcnn_fake_quant_model.prototxt`: quantized model file for accuracy simulation in the Caffe environment.
  - `faster_rcnn_fake_quant_weights.caffemodel`: weight file of the quantized model for accuracy simulation in the Caffe environment.
  - `faster_rcnn_quant.json`: quantization information file (named after the quantized model). This file gives the node mapping between the quantized model and the original model and is used for accuracy comparison between the quantized model and the original model.

When a model is re-quantized, the existing result files will be overwritten.

4. (Optional) Convert the quantized deployable model into an offline model adapted to the Ascend AI Processor by referring to **ATC Instructions**.

### 3.3.2.2.3 Model Accuracy Analysis

Inference and quantization calibration in 3.3.2.2.2 **Quantization Example** are performed based on the built-in image dataset. Therefore, the quantization result is used only to verify whether the model is successfully quantized and cannot be used to validate the model accuracy after quantization. This section describes how to compare the model accuracy before and after quantization based on the benchmark VOC2007 dataset.

Add the `with_benchmark` argument during environment initialization to download the benchmark VOC2007 dataset.

**Preparations**

Run the following command to initialize the environment information for downloading the VOC2007 benchmark dataset:

```
bash init_env.sh CPU **/caffe-master with_benchmark
```

Alternatively,
```
bash init_env.sh CPU **/caffe-master python3.7.5 /usr/include/python3.7m with_benchmark
```

After the environment is initialized, the VOCdevkit dataset file is generated in the `amct_caffe_faster_rcnn_sample/datasets` directory in addition to the files regenerated in **Environment Initialization**.
If the `with_benchmark` argument is added during environment initialization, all subsequent quantization operations are performed based on the benchmark VOC2007 dataset.

**NOTE**

- In CPU mode, only the `[--cpu]` argument can be used in the quantization command.
- In GPU mode, the `[--gpu GPU_ID]` or `[--cpu]` argument can be used in the quantization command.

Select the argument for environment initialization as required.

**Procedure**

1. Test the accuracy of the original model (before quantization).
   Run the following command:
   ```bash
   python3.7.5 faster_rcnn_sample.py --model_file pre_model/faster_rcnn_test.pt --weights_file pre_model/VGG16_faster_rcnn_final.caffemodel --gpu 0 --pre_test
   ```
   
   For details about the available command-line options, see Table 3-10. If a message similar to the following is displayed, the execution is successful:
   
   `[AMCT][INFO]Run faster_rcnn without quantize success, and mAP is 0.8812724482290413`

2. Test the accuracy of the quantized model.
   ```bash
   python3.7.5 faster_rcnn_sample.py --model_file pre_model/faster_rcnn_test.pt --weights_file pre_model/VGG16_faster_rcnn_final.caffemodel --gpu 0
   ```
   
   If a message similar to the following is displayed, the model is successfully quantized. The mean average precision (mAP) result is an example only.
   
   `[AMCT][INFO]Run faster_rcnn with quantize success, and mAP is 0.8796338534980108`

3. Compare the model accuracy before and after quantization based on the tested mAP results.

### 3.3.2.3 Quantization Example Using the `convert_model` API

**Prerequisites**

- For details about how to prepare the model, image dataset, and calibration dataset, see 3.3.2.1.1 Quantization Preparations.
- Quantization factors:
  
  Upload the quantization factor record file to any directory on the Linux server as the AMCT installation user. The following uses the quantization factors of the ResNet-50 network available in the sample package for illustration convenience. For details about quantization factors, see 3.11.4 Quantization Factor Record File.

**Procedure**

1. Precheck the original network to test if it can run properly in the Caffe environment.
   
   This step is added to identify risks in advance such as dataset and model mismatch and model execution failures in the Caffe environment.

   Run the following command in the `sample/resnet50` directory to check the ResNet-50 network:
   ```bash
   python3.7.5 convert_model.py --model_file MODEL_FILE --weights_file WEIGHTS_FILE --record_file RECORD_FILE [--gpu GPU_ID] [--cpu][--iterations ITERATIONS] --caffe_dir CAFFE_DIR [--pre_test]
   ```
Where, **--record_file RECORD_FILE** is a required option indicating the path of the quantization factor record file (.txt). For details about the rest options, see **Table 3-10**.

An example is as follows:

```bash
python3.7.5 convert_model.py --model_file pre_model/ResNet-50-deploy.prototxt --weights_file pre_model/ResNet-50-model.caffemodel --record_file pre_model/record.txt --gpu 0 --caffe_dir caffe-master --pre_test
```

If a message similar to the following is displayed, the original model runs properly in the Caffe environment.

```
[AMCT][INFO]Run ResNet-50 without quantize success!
```

2. Run the quantization script.

```bash
python3.7.5 convert_model.py --model_file pre_model/ResNet-50-deploy.prototxt --weights_file pre_model/ResNet-50-model.caffemodel --record_file pre_model/record.txt --gpu 0 --caffe_dir caffe-master
```

If messages similar to the following are displayed, the model is successfully quantized. (The top 1 and top 5 inference accuracy results are examples only.)

```
*****final top1:0.86875
*****final top5:0.95625 //Top 1 and top 5 inference accuracy results of the quantized fake_quant model in the Caffe environment.
[AMCT][INFO]Run ResNet-50 with quantize success!
```

3. View the quantization result.

After the quantization is complete, the inference result using the accuracy simulation model obtained after quantization is displayed. Find the following files generated in the directory of the quantization script:

- **amct_log/amct_caffe.log**: AMCT log file.
- **results/convert_results**: quantization result folder, including the quantized model file as well as its weight file and quantization information file.

  - **ResNet50_deploy_model.prototxt**: quantized model file to be deployed on the Ascend AI Processor.
  - **ResNet50_deploy_weights.caffemodel**: weight file of the quantized model to be deployed on the Ascend AI Processor.
  - **ResNet50_fake_quant_model.prototxt**: quantized model file for accuracy simulation in the Caffe environment.
  - **ResNet50_fake_quant_weights.caffemodel**: weight file of the quantized model file for accuracy simulation in the Caffe environment.
  - **ResNet50_quant.json**: quantization information file (named after the quantized model). This file gives the node mapping between the quantized model and the original model and is used for accuracy comparison between the quantized model and the original model. When a model is re-quantized, the existing result files will be overwritten.

### 3.3.2.4 Quantizing the MNIST Network

This model is used to quickly verify the AMCT quantization functionality. Inference and quantization calibration are performed based on the benchmark MNIST dataset. Compare the model accuracy before and after quantization based on the tested accuracy results.
3.3.2.4.1 Quantization Preparations

Model

This section uses the MNIST network available in the sample package as an example.

Image Dataset

- If the server installed with AMCT has Internet access:
  Create the `mnist_data` directory in `sample/mnist/` on the server, switch to the directory, and run the following commands to obtain image dataset and label files:
  ```
  wget http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
  wget http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
  ```
- If the user environment has no Internet access:
  On a server with Internet access, download the corresponding software packages from the following links and upload them to the `sample/mnist/mnist_data` directory:
  b. MNIST label file: available at [http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz). The MNIST test label `t10k-labels-idx1-ubyte` is generated in the `mnist/mnist_data/` directory after the environment is initialized.

Since MNIST is a basic network, the environment initialization procedure and the quantization procedure are combined. The MNIST dataset in LMDB format is automatically generated in the `mnist/mnist_test_lmdb` directory based on the preceding files in 3.3.2.4.2 Quantization Example phase.

Calibration Dataset

For details, see Calibration Dataset.

3.3.2.4.2 Quantization Example

Procedure

1. Go to the `sample/mnist` directory and run the following command to quantize the MNIST network:
   ```
   python3.7.5 mnist_sample.py --model_file pre_model/mnist-deploy.prototxt --weights_file pre_model/mnist-model.caffemodel --gpu 0 --caffe_dir caffe-master
   ```
   For details about the available command-line options, see Table 3-10.
   If messages similar to the following are displayed, the model is successfully quantized. (The inference accuracy results are examples only.)
   ```
   *****final top1:0.9853125 //Inference accuracy of the quantized fake_quant model in the Caffe environment
   [AMCT][INFO] mnist top1 before quantize is 0.98515625, after quantize is 0.9853125 //Accuracy test results
   [AMCT][INFO]Run mnist sample with quantize success!
   ```
2. After the quantization is successful, the inference results of the accuracy simulation model and the accuracy test results before and after quantization are displayed. Find the following files generated in the directory of the quantization script:

- **amct_log/amct_caffe.log**: AMCT log file.
- **tmp**: quantization temporary folder, containing:
  - **config.json**: configuration file that describes how to quantize each layer in the model. If a quantization configuration file already exists in the directory of the quantization script, the existing quantization configuration file is overwritten by a new one with the same name in a call to `create_quant_config`. If not that case, a new quantization configuration file is created. If the accuracy of model inference drops significantly after quantization, you can modify the **config.json** file by referring to **3.3.3 Quantization Configuration**.
  - **modified_model.prototxt** and **modified_model.caffemodel**: intermediate model files.
  - **record.txt**: file that records quantization factors. For details about the prototype definition of the file, see **3.11.4 Quantization Factor Record File**.
  - **mnist_data** and **mnist_test_lmdb**: dataset folders.
- **results**: quantization result folder, containing the quantized model file and its weight file as well as the quantization information file.
  - **mnist_deploy_model.prototxt**: quantized model file to be deployed on the Ascend AI Processor.
  - **mnist_deploy_weights.caffemodel**: weight file of the quantized model to be deployed on the Ascend AI Processor.
  - **mnist_fake_quant_model.prototxt**: quantized model file for accuracy simulation in the Caffe environment.
  - **mnist_fake_quant_weights.caffemodel**: weight file of the quantized model file for accuracy simulation in the Caffe environment.
  - **mnist_quant.json**: quantization information file (named after the quantized model). This file gives the node mapping between the quantized model and the original model and is used for accuracy comparison between the quantized model and the original model.

When a model is re-quantized, the existing result files will be overwritten.

### 3.3.3 Quantization Configuration

This section describes the quantization configuration file of image classification networks.

#### 3.3.3.1 Quantization Configuration Files

If inference based on the **config.json** post-training quantization configuration file generated by the call to `create_quant_config` has significant accuracy drop, tune
the `config.json` file until the accuracy is as expected. The following is an example of the file content. Keep the layer names unique in the JSON file.

- Uniform quantization configuration file

```json
{
    "version":1,
    "batch_num":2,
    "activation_offset":true,
    "joint_quant":false,
    "do_fusion":true,
    "skip_fusion_layers":[]

    "conv1":{
        "quant_enable":true,
        "activation_quant_params":{
            "max_percentile":0.999999,
            "min_percentile":0.999999,
            "search_range":{
                0.7,
                1.3
            },
            "search_step":0.01
        },
        "weight_quant_params":{
            "wts_algo":"arq_quantize",
            "channel_wise":true
        }
    },
    "conv2":{
        "quant_enable":true,
        "activation_quant_params":{
            "max_percentile":0.999999,
            "min_percentile":0.999999,
            "search_range":{
                0.7,
                1.3
            },
            "search_step":0.01
        },
        "weight_quant_params":{
            "wts_algo":"arq_quantize",
            "channel_wise":false
        }
    }
}
```

### 3.3.3.2 Configuration File Options

The following tables describe the parameters in the configuration file.

<table>
<thead>
<tr>
<th>Table 3-13 version</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
</tr>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
</tr>
</tbody>
</table>
### Table 3-14 batch_num

<table>
<thead>
<tr>
<th>Function</th>
<th>Batch count for quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>Greater than 0</td>
</tr>
<tr>
<td>Description</td>
<td>If this option is not set, the default value 1 is used. It is recommended that the number of images in the calibration dataset be less than or equal to 50. The value of <code>batch_num</code> is calculated based on the value of <code>batch_size</code>. <code>batch_num x batch_size</code> equals the number of images in the calibration dataset used for quantization. <code>batch_size</code> indicates the number of images per batch.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>1</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 3-15 activation_offset

<table>
<thead>
<tr>
<th>Function</th>
<th>Symmetric quantization or asymmetric quantization select for activation quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td>If it is set to true, asymmetric quantization is used. If it is set to false, symmetric quantization is used.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>true</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 3-16 joint_quant

<table>
<thead>
<tr>
<th>Function</th>
<th>Eltwise joint quantization switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td>If it is set to true, Eltwise joint quantization is enabled. If it is set to false, Eltwise joint quantization function is disabled.</td>
</tr>
</tbody>
</table>
Table 3-17 do_fusion

<table>
<thead>
<tr>
<th>Function</th>
<th>BN fusion switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td>If it is set to true, BN fusion is enabled. If it is set to false, BN fusion is disabled.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>true</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 3-18 skip_fusion_layers

<table>
<thead>
<tr>
<th>Function</th>
<th>Layer skip in BN fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value Range</td>
<td>Layers to skip in BN fusion. For details about the supported fusion patterns, see 3.1.2.4 Fusion Support.</td>
</tr>
<tr>
<td>Description</td>
<td>Layers to skip in BN fusion</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>-</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 3-19 layer_config

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization configuration of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
</tbody>
</table>
| Description | Includes the following parameters:  
|             | ● quant_enable  
|             | ● activation_quant_params  
|             | ● weight_quant_params |
| Recommended Value | None |
| Required/Optional | Optional |

**Table 3-20 quant_enable**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization enable by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td>If it is set to true, the layer is to be quantized. If it is set to false, otherwise.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>true</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 3-21 activation_quant_params**

<table>
<thead>
<tr>
<th>Function</th>
<th>Activation quantization parameters of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
</tbody>
</table>
| Description | Includes the following parameters:  
|             | ● max_percentile  
|             | ● min_percentile  
|             | ● search_range  
|             | ● search_step |
| Recommended Value | None |
| Required/Optional | Optional |
### Table 3-22 weight_quant_params

<table>
<thead>
<tr>
<th>Function</th>
<th>Weight quantization parameters of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>Includes the following parameters in uniform quantization:</td>
</tr>
<tr>
<td></td>
<td>● wts_algo</td>
</tr>
<tr>
<td></td>
<td>● channel_wise</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>None</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 3-23 max_percentile

<table>
<thead>
<tr>
<th>Function</th>
<th>Upper search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0.5, 1]</td>
</tr>
<tr>
<td>Description</td>
<td>Indicates the maximum number to be considered as the search result among a group of numbers in descending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the maximum, that is, the largest number. A larger value indicates that the upper clip limit is closer to the maximum value of the data to be quantized.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>0.999999</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 3-24 min_percentile

<table>
<thead>
<tr>
<th>Function</th>
<th>Lower search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0.5, 1]</td>
</tr>
</tbody>
</table>
Indicates the minimum number to be considered as the search result among a group of numbers in ascending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the minimum, that is, the smallest number. A larger value indicates that the lower clip limit is closer to the minimum value of the data to be quantized.

<table>
<thead>
<tr>
<th>Recommended Value</th>
<th>0.999999</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 3-25 search_range

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search range: [search_range_start, search_range_end]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>A list of two floats</td>
</tr>
<tr>
<td>Value Range</td>
<td>0 &lt; search_range_start &lt; search_range_end</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the quantization factor search range:</td>
</tr>
<tr>
<td></td>
<td>● search_range_start: search start.</td>
</tr>
<tr>
<td></td>
<td>● search_range_end: search end.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>[0.7, 1.3]</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 3-26 search_step

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0, (search_range_end – search_range_start)]</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the quantization factor search step. A smaller value indicates a smaller step. The number of search iterations is specified by search_iteration = (search_range_end – search_range_start)/search_step. The search becomes time-consuming with the increase of the number of search iterations. In this scenario, processes may be suspended.</td>
</tr>
</tbody>
</table>
### Table 3-27 wts_algo

<table>
<thead>
<tr>
<th>Function</th>
<th>Weight quantization algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value Range</td>
<td>arq_quantize</td>
</tr>
<tr>
<td>Description</td>
<td>arq_quantize: uniform quantization</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>None</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 3-28 channel_wise

<table>
<thead>
<tr>
<th>Function</th>
<th>Whether to use different quantization factors for each channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
</tbody>
</table>
| Description    | • If it is set to true, channels are separately quantized using different quantization factors.  
                | • If it is set to false, all channels are quantized altogether using the same quantization factors. |
| Recommended Value | true                                                               |
| Required/Optional | Optional                                                           |

### 3.3.3.3 Configuration Tuning

If the inference accuracy of the model quantized based on the default configuration in the `config.json` file drops significantly, perform the following steps to tune the post-training quantization configuration file: The following provides instructions on manually tuning the configuration file. To use the automatic tuning feature provided by AMCT, see **3.5 Accuracy-oriented Automatic Quantization Rollback**.
Step 1 Execute the quantization script in the amct_caffe_sample.tar.gz package to perform quantization based on the default configuration generated by the 3.9.2.1 create_quant_config API.

Step 2 If the inference accuracy with the model quantized in Step 1 is as expected, configuration tuning ends. Otherwise, go to Step 3.

Step 3 Tune batch_num in the quantization configuration file.

batch_num controls the batch count for quantization. Tune it based on the batch size and the number of images required for quantization. Generally, a larger quantity of data samples used in a quantization process indicates a smaller accuracy loss after quantization. However, excessive data does not improve accuracy, but occupies more memory and reduces the quantization speed, hence resulting in insufficient memory, video RAM, and thread resources. Therefore, it is recommended that the product of batch_num and batch_size be 16 or 32.

Step 4 If the inference accuracy with the model quantized in Step 3 is as expected, configuration tuning ends. Otherwise, go to Step 5.

Step 5 Tune quant_enable in the quantization configuration file.

quant_enable specifies whether to quantize a layer. If set to true, the layer is to be quantized. If set to false, otherwise. If the configuration of a layer is not present, the quantization of the layer is skipped. When the network accuracy is not as expected, locate the quantization-sensitive layers (whose error increases significantly after quantization) in the network, and disable quantization on these layers as needed. There are two methods to identify the quantization sensitive layer. One is based on the model structure. Generally, the accuracy of the first layer, last layer, and layers with a small number of parameters on the network decreases greatly after quantization. In the other method, the accuracy comparison tool can be used to compare the output errors of the original model and the quantized model layer by layer (for example, the cosine similarity is used as a standard, and the similarity must be greater than 0.99). If a layer with a large error is found, the layer is preferentially rolled back.

Step 6 If the inference accuracy with the model quantized in Step 5 is as expected, configuration tuning ends. Otherwise, go to Step 7.

Step 7 Tune the values of activation_quant_params and weight_quant_params in the quantization configuration file.

- Data is clipped to the range [left,right] specified by the activation_quant_params parameters. Generally, values distributed near a boundary are sparse, and clip may be performed on all the values, to improve the accuracy. A larger value of min_percentile (max_percentile) indicates that left (right) is closer to the minimum value (maximum value) of the to-be-quantized data. search_range and search_step affect the range of [left, right]. Generally, a larger value of search_range and a smaller value of search_step may achieve higher quantization accuracy, but the quantization takes more time.

- channel_wise in weight_quant_params determines whether to use a different quantization factor for each channel during weight quantization. If set to true, channels are separately quantized using different quantization factors. If set to false, all channels are quantized altogether using the same quantization factors. Generally, the inference accuracy is higher if the channels are separately quantized. However, InnerProduct and AVE Pooling
layers are channel-irrelevant. When `channel_wise` is set to `True`, an error is reported.

**Step 8** If the inference accuracy with the model quantized in **Step 7** is as expected, configuration tuning ends. Otherwise, it indicates that quantization has severe adverse impact on the inference accuracy. In this case, remove the quantization configuration.

----End
3.4 Quantization Aware Training
3.4.1 Quantization Example

Prerequisites

- **Model**
  Upload a Caffe model and its weight file to any directory on the Linux server as the AMCT installation user.
  The following uses the `ResNet-50_retrain.prototxt` model file downloaded by running `sample/resnet50/download_prototxt.py` as an example.

- **Image dataset**
  Because quantization aware training needs huge data to further optimize quantization parameters, the dataset used for retrain is the ImageNet dataset in LMDB format. For details about how to prepare a dataset, see the `caffe-master/examples/imagenet/readme.md` file of the Caffe project.

- **Calibration dataset**
  To ensure the quantization accuracy, the source of the calibration dataset must be the same as that of the validation dataset.

Procedure

1. Download the model file.

   Go to the `sample/resnet50` directory and run the following command to download the `ResNet-50_retrain.prototxt` model file:
   ```
   python3.7.5 download_prototxt.py --caffe_dir CAFFE_DIR --close_certificate_verify
   ```

   *Table 3-29* describes the available command-line options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--h</td>
<td>(Optional) Displays help information.</td>
</tr>
<tr>
<td>--caffe_dir CAFFE_DIR</td>
<td>(Required) Sets the directory of the Caffe source code, which can be relative or absolute.</td>
</tr>
<tr>
<td>--close_certificate_verify</td>
<td>(Optional) Disables certificate validation to ensure successful download.</td>
</tr>
<tr>
<td></td>
<td>If a message is displayed indicating that authentication fails during model download, include this option to your download command and try again.</td>
</tr>
</tbody>
</table>

An example is as follows:

```python3.7.5 download_prototxt.py --caffe_dir caffe-master --close_certificate_verify```

If messages similar to the following are displayed, the model file is successfully downloaded:

```
```
You can view the downloaded model in the sample/resnet50/pre_model directory as prompted. ResNet-50-deploy.prototxt is used in the post-training quantization scenario. For details, see 3.3.2.1 Quantizing an Image Classification Network.


Go to the sample/resnet50 directory and run the following command to run quantization aware training on the ResNet-50 model:

```
python3.7.5 ResNet50_retrain.py --model_file MODEL_FILE --weights_file WEIGHTS_FILE [--gpu GPU_ID] [--cpu] --caffe_dir CAFFE_DIR --train_data TRAIN_DATA --test_data TEST_DATA
```

For details about the available command-line options, see Table 3-10 and the following table.

### Table 3-30 Command-line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--train_data TRAIN_DATA</td>
<td>● Required. ● A string. ● Default: None ● Dataset used for quantization aware training.</td>
</tr>
<tr>
<td>--test_data TEST_DATA</td>
<td>● Required. ● A string. ● Default: None ● Dataset used to test the accuracy during retrain.</td>
</tr>
<tr>
<td>--train_batch TRAIN_BATCH</td>
<td>● Optional. ● An int. ● Default: 32 ● Batch size of the trained network. NOTE Do not set batch_size too large to avoid video memory/RAM insufficiency during quantization.</td>
</tr>
<tr>
<td>--train_iter TRAIN_ITER</td>
<td>● Optional. ● An int. ● Default: 1000 ● Number of training iterations.</td>
</tr>
<tr>
<td>--test_batch TEST_BATCH</td>
<td>● Optional. ● An int. ● Default: 1 ● Batch size of the test network.</td>
</tr>
</tbody>
</table>
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--test_iter TEST_ITER</td>
<td>● Optional.</td>
</tr>
<tr>
<td></td>
<td>● An int.</td>
</tr>
<tr>
<td></td>
<td>● Default: 500</td>
</tr>
<tr>
<td></td>
<td>● Number of test iterations.</td>
</tr>
</tbody>
</table>

**NOTE**
The ImageNet dataset has only 50,000 images. Therefore, keep `test_batch * test_iter` within 50,000.

An example is as follows:

```bash
```

If messages similar to the following are displayed, the quantization aware training is successful:

```
Network initialization done.
...
Top 1 accuracy = 0.688
Top 5 accuracy = 0.934
```

3. **View the result.**

After the quantization aware training is complete, the following files are generated in the `sample/resnet50` directory. (When quantization aware training is run again on the model, the existing result files will be overwritten.)

- `amct_log`: AMCT log folder, including the quantization aware training log `amct_caffe.log`.
- `results/retrain_results`: result folder, containing the quantized model file and its weight file.
  - `retrain_atc_model.prototxt`: quantized model file that can be deployed on the Ascend AI Processor. This file can be converted to an offline model using the ATC tool.
  - `retrain_deploy_model.prototxt`: quantized model file that can be deployed on the Ascend AI Processor. To convert this file to an offline model using the ATC tool, refer to 4 to edit the file.
  - `retrain_deploy_weights.caffemodel`: weight file of the quantized model to be deployed on the Ascend AI Processor.
  - `retrain_fake_quant_model.prototxt`: quantized model file for accuracy simulation in the Caffe environment.
  - `retrain_fake_quant_weights.caffemodel`: weight file of the quantized model for accuracy simulation in the Caffe environment.
The `retrain_atc_model.prototxt` model file is generated only after the ResNet-50 sample network is retrained, which is not a result of AMCT. To create a similar file of another network model, refer to 4.

The `retrain_deploy_model.prototxt` model file cannot be directly converted to an offline model by using the ATC tool, because it contains layers that are not supported by the ATC tool. The quantization aware training script of the ResNet-50 sample automatically deletes the unsupported layers, so that you can generate an offline model based on `retrain_atc_model.prototxt` and `retrain_deploy_weights.caffemodel` by using the ATC tool. For details, see ATC Instructions.

- **tmp**: quantization aware training temporary directory, containing:
  - config.json: quantization aware training configuration file that describes how to train each layer in the model. If a quantization aware training configuration file already exists in the directory where the quantization aware training script is located, when [3.9.3.1] create_quant_retrain_config is called again, the existing configuration file is overwritten if the name of the new configuration file is the same as that of the existing file. Otherwise, a new configuration file is generated.
  - Solving the accuracy of model inference is not as expected after quantization aware training, increase the iterations.
  - modified_model.prototxt and modified_model.caffemodel: intermediate model files
  - record.txt: file that records quantization factors. For details about the prototype definition of the file, see 3.11.4 Quantization Factor Record File.
  - solver.prototxt: quantization aware training configuration file; solver_iter_10.caffemodel and solver_iter_10.solverstate: model snapshots generated after quantization aware training.

Add the test phase to the solver in the `ResNet50_retrain.py` script. The sample code is as follows.

```python
def train(model_file: str, weights_file: str, solver_file: str):
    s = caffe.proto.caffe_pb2.SolverParameter()
    s.net = model_file
    s.lr_policy = 'step'
    s.base_lr = 0.0001
    s.stepsize = 10
    s.gamma = 0.1
    s.momentum = 0.9
    s.weight_decay = 0.0001
    s.test_initialization = False
    s.max_iter = args.train_iter
    s.test_interval = args.train_iter
    s.test_iter[:] = [args.test_iter]
    s.snapshot = args.train_iter
    with open(solver_file, 'w') as f:
        f.write(str(s))
    solver = caffe.SGDSolver(solver_file)
    solver.net.copy_from(weights_file)
    solver.solve()
```
Corresponding parameters are contained in the generated `solver.prototxt` file through the above code. The file template is as follows.

- test_iter: 1
- test_interval: 4
- base_lr: 9.999999747378752e-05
- max_iter: 4
- lr_policy: "step"
- gamma: 0.10000000149011612
- momentum: 0.8999999761581421
- weight_decay: 9.999999747378752e-05
- stepsize: 10
- snapshot: 4
- net: 
  - "$HOME/amct_path/sample/resnet50/tmp/modified_model.prototxt"
- test_initialization: false

The parameters are described as follows:

- **test_iter**: (repeated) number of test iterations. Must be greater than or equal to `batch_num` of the shift factor N. Otherwise, the shift factor N fails to be calculated due to data insufficiency.
- **test_interval**: interval between tests (in training iterations). Defaults to 0. You are advised to set this parameter to the `max_iter` factor (`test_interval==max_iter` in the sample). That is, only one test is performed after the training phase.
- **max_iter**: number of training iterations.
- **net**: model to train. For Caffe, one `net` can be configured for training and test separately (different operators are executed in different phases). You can also specify the model to train and model to test by specifying `train_net` and `test_net`, respectively. Since AMCT generates only one model (different operators are executed in different phases), only `net` is supported and `train_net` and `test_net` are not configurable.
- **test_initialization**: a bool. If it is set to `True` (default), the original model will be prechecked before training. In this case, however, the parameters are all initialized to 0 resulting in a shift factor N error. Therefore, the precheck needs to be disabled. That is, set `test_initialization` to `False`.

4. (Optional) Modify the generated deployable model.

   If you have run quantization aware training on a network other than the ResNet-50 sample network and you need to convert the generated deployable model file into an offline model adapted to the Ascend AI Processor, modify the model file as follows. For details about the layers supported by the ATC tool, see "Operator Specifications > Caffe Operator Specifications" in ATC Instructions.

   a. Check if there are layers unsupported by the ATC tool in the deployable model.

   b. Take the ResNet-50 sample as an example. Replace the Data layers in the deployable model to input layers and remove the Accuracy and SoftmaxWithLoss output layers. See the following figure.
An example of the modified code is as follows.

```plaintext
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param {
    shape {
      dim: 1
      dim: 3
      dim: 224
      dim: 224
    }
  }
}
```

### 3.4.2 Quantization Configuration

#### 3.4.2.1 Overview

If inference based on the `config.json` quantization aware training configuration file generated by the [3.9.3.1 create_quant_retrain_config](#) call has significant accuracy drop, tune the `config.json` file until the accuracy is as expected. The following is an example of the file content. Keep the layer names unique in the JSON file.

```json
{
  "version": 1,
  "conv1": {
    "retrain_enable": true,
    "retrain_data_config": {
      "algo": "ulq_quantize"
    },
    "retrain_weight_config": {
      "algo": "arq_retrain",
      "channel_wise": true
    }
  },
  "conv2_1/expand": {
    "retrain_enable": true,
    "retrain_data_config": {
      "algo": "ulq_quantize"
    },
    "retrain_weight_config": {
      "algo": "arq_retrain",
      "channel_wise": true
    }
  },
  "conv2_1/dwise": {
    "retrain_enable": true,
    "retrain_data_config": {
      "algo": "arq_retrain",
      "channel_wise": true
    }
  }
}
```
3.4.2.2 Configuration File Options

The following describes the configuration options available in the configuration file. Note that Table 3-37 to Table 3-39 are available only when you manually tune the quantization configuration file.

**Table 3-31 version**

<table>
<thead>
<tr>
<th>Function</th>
<th>Version number of the quantization configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>Currently, only version 1 is available.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>1</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 3-32 retrain_enable**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization aware training enable by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td>If set to true, quantization aware training is performed at this layer. If set to false, otherwise.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>true</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 3-33 retrain_data_config**

<table>
<thead>
<tr>
<th>Function</th>
<th>Activation quantization configuration by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>Includes the following parameters:</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>algo</td>
<td>selects the quantization algorithm, default to <em>ulq_quantize</em>.</td>
</tr>
<tr>
<td>clip_max</td>
<td>sets the upper limit of the clip quantization algorithm, default to be empty.</td>
</tr>
<tr>
<td>clip_min</td>
<td>sets the lower limit of the clip quantization algorithm, default to be empty.</td>
</tr>
<tr>
<td>fixed_min</td>
<td>fixes the minimum value of the clip quantization algorithm at 0, default to be empty.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommended Value</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 3-34  retrain_weight_config**

<table>
<thead>
<tr>
<th>Function</th>
<th>Weight quantization configuration by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Includes the following parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>algo</td>
<td>quantization algorithm select, default to <em>arq_retrain</em>.</td>
</tr>
<tr>
<td>channel_wise</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommended Value</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 3-35  algo**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization algorithm by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ulq_quantize</td>
<td>ULQ clip quantization algorithm.</td>
</tr>
<tr>
<td>arq_retrain</td>
<td>ARQ quantization algorithm.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommended Value</th>
<th>Set to <em>ulq_quantize</em> for activation quantization or <em>arq_retrain</em> for weight quantization.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>
### Table 3-36 channel_wise

<table>
<thead>
<tr>
<th>Function</th>
<th>Whether to use different quantization factors for each channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• If set to <strong>true</strong>, channels are separately quantized using different quantization factors.</td>
</tr>
<tr>
<td></td>
<td>• If set to <strong>false</strong>, all channels are quantized altogether using the same quantization factors.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>true</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 3-37 fixed_min

<table>
<thead>
<tr>
<th>Function</th>
<th>Lower limit enable of the activation quantization algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• If set to <strong>true</strong>, the lower limit of the activation quantization algorithm is fixed at 0.</td>
</tr>
<tr>
<td></td>
<td>• If set to <strong>false</strong>, the lower limit of the activation quantization algorithm is not fixed.</td>
</tr>
<tr>
<td></td>
<td>If this option is not included, the AMCT automatically sets the lower limit of the activation quantization algorithm according to the graph structure.</td>
</tr>
<tr>
<td></td>
<td>If this option is included: when the upstream layer of the quantization layer is ReLU, you need to manually set this option to <strong>true</strong>; when the upstream layer of the quantization layer is not ReLU, you need to manually set this option to <strong>false</strong>.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>Do not include this option.</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 3-38 clip_max

<table>
<thead>
<tr>
<th>Function</th>
<th>Upper limit of the activation quantization algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
</tbody>
</table>
### 3.4.2.3 Configuration Tuning

If the inference accuracy of the model quantized based on the default `config.json` configuration drops significantly, perform the following steps to tune the quantization configuration file. The following provides instructions on manually tuning the configuration file. To use the automatic tuning feature provided by AMCT, see 3.5 Accuracy-oriented Automatic Quantization Rollback.

1. Execute the quantization script in the `amct_caffe_sample.tar.gz` package to perform quantization based on the default configuration generated by the `create_quant_retrain_config` call. If the quantization accuracy is as expected, the configuration tuning ends. Otherwise, go to 2.
2. Cancel the quantization of certain layers by changing the value of `retrain_enable` to `false`. Generally, the input and output layers of a model
have a greater impact on the inference result. Therefore, you can try to cancel the quantization of the input and output layers first.

If you have desirable settings for \texttt{clip\_max} and \texttt{clip\_min}, modify the quantization configuration file as follows.

```json
{
    "version": 1,
    "layername 1": {
        "retrain_enable": true,
        "retrain_data_config": {
            "algo": "ulq_quantize",
            "clip_max": 3.0,
            "clip_min": -3.0
        },
        "retrain_weight_config": {
            "algo": "arq_retrain",
            "channel_wise": true
        }
    },
    "layername 2": {
        "retrain_enable": true,
        "retrain_data_config": {
            "algo": "ulq_quantize",
            "clip_max": 3.0,
            "clip_min": -3.0
        },
        "retrain_weight_config": {
            "algo": "arq_retrain",
            "channel_wise": true
        }
    }
}
```

3. Configuration tuning ends if the inference accuracy meets the requirement. Otherwise, it indicates that quantization aware training has severe adverse impact on the inference accuracy. In this case, remove the quantization aware training configuration.

### 3.5 Accuracy-oriented Automatic Quantization Rollback

#### 3.5.1 Prerequisites

**Quantization Code Modification**

Manually implement a class inherited from \texttt{AutoCalibrationEvaluatorBase} and rewrite the three methods under the base class.

1. Implement the \texttt{evaluate} method inherited from the \texttt{model_evaluator} base class. Keep the inputs and return of the function consistent with those of the base class. The function should return a unique model accuracy metric, for example, top 1 accuracy results for image classification networks.

2. Implement the \texttt{calibration} method inherited from the \texttt{model_evaluator} base class. Keep the inputs and return of the function consistent with those of the base class. This method is used to run model inference during the calibration process. Keep the number of inference batches the same as the \texttt{batch_num} argument.
3. Implement the `metric_eval` method inherited from the `model_evaluator` base class. Keep the inputs and return of the function consistent with those of the base class. This method is used to test the accuracy of the quantized model during automatic quantization configuration search. It returns a tuple, where the first element indicates whether the accuracy is acceptable and the second element indicates the accuracy drop.

The sample code is provided as follows. Find the code file in `amct_caffe/common/auto_calibration/auto_calibration_evaluator_base.py` in the AMCT installation path.

class AutoCalibrationEvaluatorBase:
    """ the base class for ModelEvaluator"""
    def __init__(self):
        """__init__ function of class"""
        pass

def calibration(self, model): # pylint: disable=R0201
    """do the calibration"""
    Parameter:
    if framework is caffe:
        model: the prototxt model define file of caffe model
        weights: the binary caffemodel file of caffe model
    if framework is tensorflow:
        model: the graph of model
        outputs (list): a list of output nodes.
    Return:
    None
    raise NotImplementedError

def evaluate(self, model): # pylint: disable=R0201
    """evaluate the input models, get the eval metric of model"""
    Parameter:
    if framework is Caffe:
        model: the prototxt model define file of caffe model
        weights: the binary caffemodel file of caffe model
    if framework is TensorFlow:
        model: the graph of model
        outputs (list): a list of output nodes.
    Return:
    metric: the eval metric of input caffe model, such as:
    top1 accuracy for classification model, mAP for
    detection model
    raise NotImplementedError

def metric_eval(self, original_metric, new_metric) -> Tuple[bool, float]: # pylint: disable=R0201
    """whether the gap between new metric and original metric
    can satisfy the requirement."""
    Parameter:
    original_metric: metric of eval original caffe model
    new_metric: metric of eval quantized caffe model
    Return:
    metric_eval (Tuple[bool, float]): A tuple of whether the
    metric is satisfied and loss function value.
    raise NotImplementede
Model

Upload a Caffe model and its weight file to any directory on the Linux server as the AMCT installation user. This section uses the image classification network MobileNetV2 available in the sample package as an example. Download the model file in advance.

Run the following command in the `sample/mobilenetV2` directory to download the model file:

```bash
python3.7.5 download_prototxt.py --caffe_dir CAFFE_DIR --close_certificate_verify
```

The following table describes the command-line options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--h</td>
<td>(Optional) Displays help information.</td>
</tr>
<tr>
<td>--caffe_dir CAFFE_DIR</td>
<td>(Required) Sets the directory of the Caffe source code, which can be relative or absolute.</td>
</tr>
<tr>
<td>--close_certificate_verify</td>
<td>(Optional) Disables certificate validation to ensure successful download. If a message is displayed indicating that authentication fails during model download, include this option to your download command and try again.</td>
</tr>
</tbody>
</table>

An example is as follows:

```bash
python3.7.5 download_prototxt.py --caffe_dir caffe-master --close_certificate_verify
```

If messages similar to the following are displayed, the model file is successfully downloaded.

```
[INFO]Download 'mobilenet_deploy.prototxt' to '$HOME/amct/amct_caffe/sample/mobilenetV2/pre_model/mobilenet_v2_deploy.prototxt' success.
[INFO]Download file_name to '$HOME/amct/amct_caffe/sample/mobilenetV2/pre_model/mobilenet_v2.caffemodel' success.
```

You can view the downloaded model in the `sample/mobilenetV2/pre_model` directory as prompted.

Image Dataset

Prepare a validation dataset for calibrating and testing the model during automatic quantization rollback. The ImageNet dataset in LMDB format is used in this example. For details about how to prepare a dataset, see the `caffe-master/examples/imagenet/readme.md` file of the Caffe project.

If you choose to prepare your own dataset, you need to modify the data preprocessing part in the sample code to match the model inputs.
3.5.2 Rollback Example

1. Run the quantization script.
   Go to the `sample/mobilenetV2` directory and run the following command to perform accuracy-oriented automatic quantization rollback. The accuracy drop limit is 0.2% by default.
   ```bash
   ```
   The command-line options are described as follows.

   **Table 3-41 Command-line options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--h</td>
<td>(Optional) Displays help information.</td>
</tr>
<tr>
<td>--model_file MODEL_FILE</td>
<td>(Required) Sets the directory of the Caffe model file (.prototxt).</td>
</tr>
<tr>
<td>--weights_file WEIGHTS_FILE</td>
<td>(Required) Sets the directory of the Caffe weight file (.caffemodel).</td>
</tr>
<tr>
<td>--gpu GPU_ID</td>
<td>(Optional) Sets the ID of the compute GPU device.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong></td>
</tr>
<tr>
<td></td>
<td>In GPU inference mode, compile the Caffe environment of the GPU version before running the quantization script.</td>
</tr>
<tr>
<td>--cpu</td>
<td>(Optional) Enables the CPU inference mode.</td>
</tr>
<tr>
<td></td>
<td><strong>[--gpu GPU_ID] and [--cpu] are mutually exclusive.</strong> [--cpu] is used by default.</td>
</tr>
<tr>
<td>--iterations ITERATIONS</td>
<td>(Optional) Sets the batch count for inference.</td>
</tr>
<tr>
<td>--caffe_dir CAFFE_DIR</td>
<td>(Required) Sets the directory of the Caffe source code, which can be relative or absolute.</td>
</tr>
<tr>
<td>--pre_test</td>
<td>(Optional) Prechecks the original model and provides the inference result if it can run properly in the Caffe environment.</td>
</tr>
<tr>
<td>--dataset DATASET</td>
<td>(Required) Sets the directory of the ImageNet dataset in LMDB format.</td>
</tr>
</tbody>
</table>

   An example is as follows.
   ```bash
   python3.7.5 auto_calibration_mobilenet_v2_sample.py --model_file pre_model/mobilenet_v2_deploy.prototxt --weights_file pre_model/mobilenet_v2.caffemodel --gpu 0 --caffe_dir caffe-master --dataset /data/Datasets/imagenet/ilsvrc12_val_lmdb
   ```

   If messages similar to the following are displayed, the quantization is successful. (The inference accuracy results and quantization time are only examples.)

   ```bash
   # Accuracy of the original model
   2021-01-06 11:24:14,809 - INFO - [AMCT]:[AMCT]: Accuracy of original model is 0.7116875
   ```
# Accuracy of the simulation model tested based on the default quantization configuration.
2021-01-06 11:24:14,810 - INFO - [AMCT]:[AMCT]: Accuracy of global quantized model is 0.70678125

# Accuracy of the simulation model generated after automatic quantization rollback.
2021-01-06 11:24:14,810 - INFO - [AMCT]:[AMCT]: Accuracy of saved model is 0.710125
2021-01-06 11:24:14,810 - INFO - [AMCT]:[AMCT]: The generated model is stored in dir: xxx/sample/mobilenetV2/results
2021-01-06 11:24:14,810 - INFO - [AMCT]:[AMCT]: The records file is stored in dir: xxx/sample/mobilenetV2/tmp

2021-01-06 11:24:14,811 - INFO - [AMCT]:[AMCT]: #>Func name: accuracy_based_auto_calibration
2021-01-06 11:24:14,811 - INFO - [AMCT]:[AMCT]: #>Cost time: 4 hours, 3 minutes, 23.26 seconds
2021-01-06 11:24:14,811 - INFO - [AMCT]:[AMCT]: ****************************************************

2. View the quantization result.

After successful automatic quantization rollback, the accuracy of the original model, the accuracy of the simulation model tested based on the default quantization configuration, and the accuracy of the simulation model generated after automatic quantization rollback are displayed. Find the following files generated in the sample/mobilenetV2 directory:

- **amct_log**:

- **config.json**: original quantization configuration file.

- **tmp/scale_offset_record.txt**: file that records quantization factors. For details about the prototype definition, see 3.11.4 Quantization Factor Record File.

- **results**:
  - `accuracy_based_auto_calibration_final_config.json`: quantization configuration file generated after accuracy-oriented automatic quantization rollback.
  - `accuracy_based_auto_calibration_ranking_information.json`: file that records the quantization sensitivity information of layers to be quantized.
  - `MobileNetV2_deploy_model.prototxt`: model file generated after automatic quantization rollback for deployment on Ascend AI Processor.
  - `MobileNetV2_deploy_weights.caffemodel`: weight file generated after automatic quantization rollback for deployment on Ascend AI Processor.
  - `MobileNetV2_fake_quant_model.prototxt`: model file generated after automatic quantization rollback for accuracy simulation in the Caffe environment.
  - `MobileNetV2_fake_quant_weights.caffemodel`: weight file generated after automatic quantization rollback for accuracy simulation in the Caffe environment.
  - `MobileNetV2_quant.json`: file that records graph fusion operations.
3.6 Tensor Decomposition

Restrictions

If a Convolution layer uses a large shape, the decomposition will be time-consuming or terminated abnormally. To prevent this problem, refer to the following before starting decomposition:

- **Reference performance specifications of the decomposition tool:**
  - CPU: Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20 GHz
  - At least 512 GB memory
  Time taken to decompose a single Convolution layer:
  - About 25s for shape (512, 512, 5, 5).
  - About 16s for shape (1024, 1024, 3, 3).
  - About 78s for shape (1024, 1024, 5, 5).
  - About 63s for shape (2048, 2048, 3, 3).
  - About 430s for shape (2048, 2048, 5, 5).

- **Memory consideration:**
  It takes about 32 GB memory to decompose the Convolution kernel with shape (2048, 2048, 5, 5).

Prerequisites

Upload the Caffe model file to be decomposed and its weight file to any directory on the Linux server. The following uses the model in `sample/resnet50/` as an example. Before using the model in this sample, download the `ResNet-50-deploy.prototxt` model file by referring to Model.

Procedure

**Step 1** Go to the amct_caffe/sample/tensor_decompose directory and execute the script for decomposing the original model:

```
python3.7.5 tensor_decomposition_sample.py --model_file MODEL_FILE --weights_file WEIGHTS_FILE --new_model_file NEW_MODEL_FILE --new_weights_file NEW_WEIGHTS_FILE --caffe_dir CAFFE_DIR
```

Table 3-42 describes the command-line options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--h</code></td>
<td>(Optional) Displays help information.</td>
</tr>
<tr>
<td><code>--model_file</code></td>
<td>(Required) Sets the directory of the Caffe model file (.prototxt).</td>
</tr>
<tr>
<td><code>--weights_file</code></td>
<td>(Required) Sets the directory of the Caffe weight file (.caffemodel).</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--new_model_file</td>
<td>(Required) Sets the directory of the result Caffe model file (.prototxt) after tensor decomposition.</td>
</tr>
<tr>
<td>NEW_MODEL_FILE</td>
<td></td>
</tr>
<tr>
<td>--new_weights_file</td>
<td>(Required) Sets the directory of the result Caffe weight file (.caffemodel) after tensor decomposition.</td>
</tr>
<tr>
<td>NEW_WEIGHTS_FILE</td>
<td></td>
</tr>
<tr>
<td>--caffe_dir</td>
<td>(Required) Sets the directory of the Caffe source code, which can be relative or absolute.</td>
</tr>
<tr>
<td>CAFFE_DIR</td>
<td></td>
</tr>
</tbody>
</table>

An example is as follows:

```
```

During the decomposition, the names of the decomposition-capable operators and the names of the decomposed operators are recorded in the log. The following log messages are examples only.

```
INFO - [AMCT]:[AMCT]: Decomposition res3a_branch2b ->['res3a_branch2b_0', 'res3a_branch2b_1', 'res3a_branch2b_2']
INFO - [AMCT]:[AMCT]: Decomposition res3b_branch2b ->['res3b_branch2b_0', 'res3b_branch2b_1', 'res3b_branch2b_2']
INFO - [AMCT]:[AMCT]: Decomposition res3c_branch2b ->['res3c_branch2b_0', 'res3c_branch2b_1', 'res3c_branch2b_2']
```

If messages similar to the following are displayed, the decomposition is successful:

```
Run tensor_decomposition success!
```

**Step 2** After the decomposition is complete, the result model and its weight file are generated to the path specified by the --new_model_file and --new_weights_file options.

```
INFO - [AMCT]:[AMCT]: The $HOME/amct/amct_caffe/sample/tensor_decompose/ResNet-50-model_tensor_decomposition.caffemodel is saved //Weight file of the result model file
INFO - [AMCT]:[AMCT]: The $HOME/amct/amct_caffe/sample/tensor_decompose/ResNet-50-deploy_tensor_decomposition.prototxt is saved //Result model file
```

**NOTE**

In normal cases, the accuracy of a decomposed model is lower than that of the original model. Therefore, fine-tuning is introduced to improve the accuracy of the decomposed model before using it for inference. Decrease the learning rate from about 0.1 times of the original learning rate. The number of epochs varies with models. The more Convolution layers are decomposed, the more epochs are required. The fine-tuned model can have accuracy improvement. However, it is also possible that the accuracy remains or even drops.

**Step 3** For details about how to quantize a decomposed model, see 3.3.2 Uniform Quantization.

----End
3.7 AMCT Update

The latest AMCT release allows you to access to the latest features. Before updating AMCT, uninstall the existing installation by referring to 3.8 AMCT Uninstallation, and then install the latest version by referring to 3.2 AMCT Installation.

After the latest AMCT release is installed, the Caffe environment needs to be rebuilt.

3.8 AMCT Uninstallation

You can uninstall the AMCT as follows:

1. Run the following command in any directory on the Linux server as the AMCT installation user:
   
   ```
   pip3.7.5 uninstall amct_caffe
   ```

2. When the following information is displayed, enter y:
   
   Uninstalling amct-caffe-(version):
   Would remove:
   ...
   Proceed (y/n)? y

   If a message similar to the following is displayed, the uninstallation is successful:
   
   Successfully uninstalled amct-caffe-(version)

   The installed Caffe will not be uninstalled during the uninstallation.

3.9 API Description

3.9.1 Common APIs

3.9.1.1 set_gpu_mode

Description

Schedules AMCT weight quantization to the GPU.

Restrictions

The GPU environment is available and CUDA 10.0 is supported. This API does not support GPU card selection. You can select a GPU card by using the CUDA environment variable (CUDA_VISIBLE_DEVICES) or by using the set_device() API of PyCaffe.

Prototype

```python
set_gpu_mode()
```
Parameters
None

Returns
None

Outputs
None

Example
import amct_caffe as amct
amct.set_gpu_mode()

NOTE
If neither 3.9.1.1 set_gpu_mode nor 3.9.1.2 set_cpu_mode is called, AMCT weight quantization is scheduled to the CPU.

3.9.1.2 set_cpu_mode

Description
Schedules AMCT weight quantization to the CPU.

Prototype

set_cpu_mode()

Parameters
None

Returns
None

Outputs
None

Example
import amct_caffe as amct
amct.set_cpu_mode()

NOTE
If neither 3.9.1.1 set_gpu_mode nor 3.9.1.2 set_cpu_mode is called, AMCT weight quantization is scheduled to the CPU.

3.9.2 Post-training Quantization
3.9.2.1 create_quant_config

Description

Applies to post-training quantization. Finds all quantization-capable layers in a graph, creates a quantization configuration file, and writes the quantization configuration information of the quantization-capable layers to the configuration file.

Restrictions

Due to data type conversion, the quantization factors in the generated quantization configuration file might be different from those in the simplified configuration file. However, the accuracy is not affected.

Prototype

create_quant_config(config_file, model_file, weights_file, skip_layers=None, batch_num=1, activation_offset=True, config_defination=None)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input / Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the quantization configuration file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The existing file (if available) in the directory will be overwritten upon this API call.</td>
<td></td>
</tr>
<tr>
<td>model_file</td>
<td>Input</td>
<td>Caffe model definition file (.prototxt).</td>
<td>A string.</td>
</tr>
<tr>
<td>weights_file</td>
<td>Input</td>
<td>Weight file of the already-trained Caffe model (.caffemodel).</td>
<td>A string.</td>
</tr>
</tbody>
</table>
| skip_layers   | Input          | Quantization-capable layers to skip.                                        | Default: None
<p>|               |                | A list of strings. Restriction: If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect. |             |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch_num</td>
<td>Input</td>
<td>Number of batches taken to generate the quantization factors.</td>
<td>An int. Value range: any integer larger than 0. Default: 1 Restrictions: • batch_num cannot be too large. The product of batch_num and batch_size equals to the number of images used during quantization. Too many images consume too much memory. • If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect.</td>
</tr>
<tr>
<td>activation_offset</td>
<td>Input</td>
<td>Whether to quantize activations with offset.</td>
<td>Default: true A bool. Restriction: If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| config_defination         | Input        | Whether to create a simplified quantization configuration file `quant.cfg` from the `calibration_config_caffe.proto` file in `/amct_caffe/proto/calibration_config_caffe.proto` under the AMCT installation path. For details about the parameters in the `calibration_config_caffe.proto` template and the generated simplified quantization configuration file `quant.cfg`, see [3.11.2 Simplified Post-training Quantization Configuration File](#).                                                                                     | Default: **None**  
A string.  
Restriction: If it is set to **None**, a configuration file is generated based on the residual arguments (`skip_layers`, `batch_num`, and `activation_offset`). Otherwise, a configuration file in JSON format is generated based on this argument. |

**Returns**

None

**Outputs**

Outputs a quantization configuration file in JSON format. (When quantization is performed again, the existing one output by this API call will be overwritten.) Assume that only layer_name1 and layer_name2 in a graph support quantization. The quantization configuration file generated by the **3.9.2.1 create_quant_config** call is as follows:

```json
{
    "version":1,
    "batch_num":2,
    "activation_offset":true,
    "joint_quant":false,
    "do_fusion":true,
    "skip_fusion_layers":[]

    "conv":{
    "quant_enable":true,
    "activation_quant_params":{
    "max_percentile":0.999999,
    "min_percentile":0.999999,
    "search_range":{
    0.7,
    1.3
    },
    "search_step":0.01
    },
    "weight_quant_params":{
    "wts_algo":"arq_quantize",
    "channel_wise":true
    }
}
```
"conv2":{
  "quant_enable":true,
  "activation_quant_params":{
    "max_percentile":0.999999,
    "min_percentile":0.999999,
    "search_range":[
      0.7,
      1.3
    ],
    "search_step":0.01
  },
  "weight_quant_params":{
    "wts_algo":"arq_quantize",
    "channel_wise":false
  }
}

Example

```python
from amct_caffe import create_quant_config

create_quant_config(config_file="./configs/config.json",
                     model_file="./pretrained_model/model.prototxt",
                     weights_file="./pretrained_model/model.caffemodel",
                     skip_layers=None,
                     batch_num=1,
                     activation_offset=True)

create_quant_config(config_file="./configs/config.json",
                     model_file="./pretrained_model/model.prototxt",
                     weights_file="./pretrained_model/model.caffemodel",
                     config_defination="./configs/quant.cfg")
```

### 3.9.2.2 init

**Description**

Applies to post-training quantization. Initializes AMCT, saves the quantization factor record file, and parses the user model into a graph that can be passed to the calls to 3.9.2.3 quantize_model and 3.9.2.4 save_model.

**Prototype**

```python
graph = init(config_file, model_file, weights_file, scale_offset_record_file)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization configuration file.</td>
<td>A string.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>model_file</td>
<td>Input</td>
<td>Caffe model definition file (.prototxt), the same as <code>model_file</code> passed to the <code>create_quant_config</code> call.</td>
<td>A string. Restriction: For layers for inference, the settings in <code>LayerParameter</code> in <code>model_file</code> must meet inference requirements. For example, <code>use_global_stats</code> of the BatchNorm layer must be set to 1.</td>
</tr>
<tr>
<td>weights_file</td>
<td>Input</td>
<td>Already-trained Caffe model file (.caffemodel), the same as <code>weights_file</code> passed to the <code>create_quant_config</code> call.</td>
<td>A string.</td>
</tr>
<tr>
<td>scale_offset_record_file</td>
<td>Input</td>
<td>Quantization factor file. The existing file in the directory will be overwritten upon this API call.</td>
<td>A string.</td>
</tr>
<tr>
<td>graph</td>
<td>Return</td>
<td>Graph parsed from the model.</td>
<td>An AMCT-defined Graph.</td>
</tr>
</tbody>
</table>

**Returns**

A graph parsed from the model.

**Outputs**

Outputs a quantization factor record file to the `scale_offset_record_file` directory by creating a file or overwriting the existing one in the directory.

When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.

**Example**

```python
from amct_caffe import init
# Initialize the tool.
graph = init(config_file="./configs/config.json",
             model_file="./pretrained_model/model.prototxt",
             weights_file="./pretrained_model/model.caffemodel",
             scale_offset_record_file="./recording.txt")
```
### 3.9.2.3 quantize_model

#### Description
Applies to post-training quantization. Quantizes a graph based on the quantization configuration file `config_file`, inserts quantization layers, and saves the new network to a file.

#### Prototype
```
quantize_model(graph, modified_model_file, modified_weights_file)
```

#### Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph</td>
<td>Input</td>
<td>Graph parsed from the user model by the <code>init</code> API.</td>
<td>An AMCT-defined Graph.</td>
</tr>
<tr>
<td>modified_model_file</td>
<td>Input</td>
<td>File name of the quantized Caffe model definition file (.prototxt).</td>
<td>A string.</td>
</tr>
<tr>
<td>modified_weights_file</td>
<td>Input</td>
<td>File name of the quantized model weight file (.caffemodel).</td>
<td>A string.</td>
</tr>
</tbody>
</table>

#### Returns
None

#### Outputs
- Outputs a quantization factor record file by writing the weight quantization factors (`scale_w` and `offset_w`) of each quantization layer to `scale_offset_record_file` passed to the `init` call.
- Outputs `modified_model_file`, the definition file of the result model inserted with quantization layers.
- Outputs `modified_weights_file`, the weight file of the result model inserted with quantization layers.

When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.

#### Example
```
from amct_caffe import quantize_model
# Quantize the model.
quantize_model(graph=graph,
               modified_model_file="./quantized_model/modified_model.prototxt",
               modified_weights_file="./quantized_model/modified_model.caffemodel")
```
3.9.2.4 save_model

Description

Applies to post-training quantization. Saves a model for accuracy simulation in the Caffe environment and/or a model for online inference running on the Ascend AI Processor.

Restrictions

- Use this API after `batch_num` is reached. Otherwise, the quantization factors are incorrect and the quantization result is compromised.
- Due to data type conversion, the quantization factors (scale and offset) in the generated quantization configuration file might be different from those in the simplified configuration file. However, the accuracy is not affected.
- `scale_offset_record_file` must contain the quantization factors of all quantization layers. Otherwise, an error is reported. That is, `modified_model_file` and `modified_weights_file` in the `quantize_model` must complete `batch_num` inference iterations in the Caffe environment.

Prototype

```
save_model(graph, save_type, save_path)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph</td>
<td>Input</td>
<td>Graph output from the <code>quantize_model</code> API call.</td>
<td>An AMCT-defined Graph.</td>
</tr>
<tr>
<td>save_type</td>
<td>Input</td>
<td>Type of the model to save:</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Fakequant</strong>: a model for accuracy simulation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Deploy</strong>: a model deployable on the Ascend AI Processor.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Both</strong>: two models, one for accuracy simulation and one deployable on</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>the Ascend AI Processor.</td>
<td></td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path.</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Must be prefixed with the model name, for example, <code>./quantized_model/*model</code></td>
<td></td>
</tr>
</tbody>
</table>

Returns

None
Outputs

- Outputs a model for accuracy simulation in the Caffe environment and its weight file, with names containing the `fake_quant` keyword.
- Outputs a deployable model file and its model weight file, with names containing the `deploy` keyword. The model can be deployed on the Ascend AI Processor after being converted by the ATC tool.
- Outputs a quantization information file that records the locations of the quantization layers inserted by AMCT and operator fusion information, which are used for accuracy testing of the quantized model.

When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.

Example

```python
from amct_caffe import save_model

# In the Caffe environment, perform batch_num inference on the modified model for quantization.
run_caffe_model(modified_model_file, modified_weights_file, batch_num)

# Insert quantization layers, and save the quantized model to a .prototxt model file and a .caffemodel weight file. The following files can be found in the ./quantized_model path:
model_fake_quant_model.prototxt, model_fake_quant_weights.caffemodel,
model_deploy_model.prototxt, model_deploy_weights.caffemodel, and model_quant.json.

save_model(graph=graph,
    save_type="Both",
    save_path="./quantized_model/model")
```

### 3.9.2.5 convert_model

**Description**

Based on the computed quantization factors, converts a Caffe model to two models, one for accuracy simulation in the Caffe environment and the other for online inference on the Ascend AI Processor.

**Restrictions**

- The user model must match the quantization factor record file. For example, if the Conv+BN+Scale composite is fused before computation to generate the quantization factors, the Conv+BN+Scale composite in the Caffe model to be converted also needs to be fused in advance.
- The format and content of the quantization factor record file must comply with the AMCT requirements defined in 3.11.4 Quantization Factor Record File.
- The quantization-capable layers include: InnerProduct (quantization not supported if `transpose = true` or `axis! = 1`), Convolution (using a 4 x 4 filter), Deconvolution (using a 1-dilated 4 x 4 filter with `group = 1`), and AVE Pooling.
- This API allows fusion of the Conv+BN+Scale composite and mode.
- Only a standard floating-point model of the Caffe framework can be passed to this call. Secondary quantization on a quantized model (inserted with a Quant, DeQuant, or AntiQuant layer or whose parameters have been quantized to the int8 or int32 data type) is not supported.
Prototype

call the model with:

```
convert_model(model_file,weights_file,scale_offset_record_file,save_path)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_file</td>
<td>Input</td>
<td>Caffe model definition file (.prototxt).</td>
<td>A string. Restriction: For layers for inference, the settings in LayerParameter in model_file must meet inference requirements. For example, use_global_stats of the BatchNorm layer must be set to 1.</td>
</tr>
<tr>
<td>weights_file</td>
<td>Input</td>
<td>Weight file of the already-trained Caffe model (.caffemodel).</td>
<td>A string.</td>
</tr>
<tr>
<td>scale_offset_record_file</td>
<td>Input</td>
<td>Computed quantization factor record file (.txt)</td>
<td>A string.</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path. The path must contain the model name prefix, for example, ./quantized_model/*.model.</td>
<td>A string.</td>
</tr>
</tbody>
</table>

Returns

None

Outputs

- Outputs a model for accuracy simulation in the Caffe environment and its weight file, with names containing the fake_quant keyword.
- Outputs a deployable model file and its model weight file, with names containing the deploy keyword. The model can be deployed on Ascend AI Processor after being converted by the ATC tool.
- Outputs a quantization information file that records the locations of the quantization layers inserted by AMCT and operator fusion information, which are used for accuracy testing of the quantized model.

When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.
Example

```python
from amct_caffe import convert_model
convert_model(model_file='ResNet-50-deploy.prototxt',
               weights_file='ResNet-50-weights.caffemodel',
               scale_offset_record_file='record.txt',
               save_path='./quantized_model/model')
```

3.9.2.6 accuracy_based_auto_calibration

**Description**

Performs automatic calibration on the input model based on the input configuration file to search for a quantization configuration that meets the accuracy requirement, resulting in a model that can serve both for accuracy simulation (**fake_quant**) in the Caffe environment and for inference (**deploy**) on the Ascend AI Processor.

**Restrictions**

None

**Prototype**

```python
accuracy_based_auto_calibration(model_file,weights_file,model_evaluator,conf
ig_file,record_file,save_dir,strategy='BinarySearch',sensitivity='CosineSimilarity')
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_file</td>
<td>Input</td>
<td>Caffe model definition file (.prototxt).</td>
<td>A string.</td>
</tr>
<tr>
<td>weights_file</td>
<td>Input</td>
<td>Weight file of the already-trained Caffe model (.caffemodel).</td>
<td>A string.</td>
</tr>
<tr>
<td>model_evaluator</td>
<td>Input</td>
<td>Python instance for automatic quantization calibration and accuracy evaluation.</td>
<td>A Python instance.</td>
</tr>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization configuration file.</td>
<td>A string.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Quantization factor record file. If the file exists, it will be overwritten.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_dir</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, ./quantized_model/model.</td>
<td>A string.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>strategy</td>
<td>Input</td>
<td>Strategy of searching for the quantization configuration that meets accuracy requirements. By default, the binary search strategy is used.</td>
<td>A string or a Python instance. Default: <code>BinarySearch</code></td>
</tr>
<tr>
<td>sensitivity</td>
<td>Input</td>
<td>Metric used to evaluate the quantization sensitivity of each quantization layer. By default, the cosine similarity metric is used.</td>
<td>A string or a Python instance. Default: <code>CosineSimilarity</code></td>
</tr>
</tbody>
</table>

**Returns**
None

**Outputs**
- Outputs a model for accuracy simulation in the Caffe environment and its weight file, with names containing the `fake_quant` keyword.
- Outputs a deployable model file and its model weight file, with names containing the `deploy` keyword. The model can be deployed on the Ascend AI Processor after being converted by the ATC tool.
- Outputs a quantization factor record file by writing the weight quantization factors (`scale_w` and `offset_w`) of each quantization layer to `record_file` passed to the call.
- Outputs a quantization information file that records the locations of the quantization layers inserted by AMCT and operator fusion information, which are used for accuracy testing of the quantized model.
- Outputs a sensitivity information file which records the sensitivity information of the layers to quantize and is used for accuracy-oriented automatic quantization rollback.
- Outputs an automatic quantization rollback history file which records information about the layers to roll back.

**Example**

```python
import amct_caffe as amct
from amct_caffe.common.auto_calibration import AutoCalibrationEvaluatorBase
from amct_caffe.common.auto_calibration import BinarySearchStrategy
from amct_caffe.common.auto_calibration import CosineSimilaritySensitivity
class AutoCalibrationEvaluator(AutoCalibrationEvaluatorBase):
    def __init__(self):
        ""
        evaluate_batch_num is the needed batch num for evaluating
        the model. Larger evaluate_batch_num is recommended, because
        the evaluation metric of input model can be more precise
        with larger eval dataset.
        ""
        super().__init__()""```
def calibration(self, model_file, weights_file):
    ""
    Function:
    do the calibration with model
    Parameter:
    model_file: the prototxt model define file of caffe model
    weights_file: the binary caffemodel file of caffe model
    ""
    run_caffe_model(args, model_file, weights_file, CALIBRATION_BATCH_NUM)

def evaluate(self, model_file, weights_file):
    ""
    Function:
    evaluate the model with batch_num of data, return the eval
    metric of the input model, such as top1 for classification
    model, mAP for detection model and so on.
    Parameter:
    model_file: the prototxt model define file of caffe model
    weights_file: the binary caffemodel file of caffe model
    ""
    return do_benchmark_test(args, model_file, weights_file, args.iterations)

def metric_eval(self, original_metric, new_metric):
    ""
    Function:
    whether the metric of new fake quant model can satisfy the
    requirement
    Parameter:
    original_metric: the metric of non quantized model
    new_metric: the metric of new quantized model
    ""
    # the loss of top1 acc needs to be less than 0.2%
    loss = original_metric - new_metric
    if loss * 100 < 0.2:
        return True, loss
    return False, loss

# step 1: create the quant config file
cfg_file = './config.json'
skip_layers = []
batch_num = CALIBRATION_BATCH_NUM
activation_offset = True
amct.create_quant_config(config_json_file, model_file, weights_file,
                          skip_layers, batch_num, activation_offset)
scale_offset_record_file = os.path.join(TMP, 'scale_offset_record.txt')
result_path = os.path.join(RESULT, 'MobileNetV2')
evaluator = AutoCalibrationEvaluator()

# step 2: start the accuracy_based_auto_calibration process
amct.accuracy_based_auto_calibration(args.model_file,
                                      args.weights_file,
                                      evaluator,
                                      config_json_file,
                                      scale_offset_record_file,
                                      result_path)

3.9.3 Quantization Aware Training
3.9.3.1 create_quant_retrain_config

Description
Applies to quantization aware training. Finds all quantization-capable layers in a graph, creates a quantization configuration file, and writes the quantization configuration information of the quantization-capable layers to the configuration file.

Restrictions
None

Prototype
create_quant_retrain_config(config_file, model_file, weights_file, config_definition=None)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the configuration file, including the file name. The existing file (if available) in the directory will be overwritten upon this API call.</td>
<td>A string.</td>
</tr>
<tr>
<td>model_file</td>
<td>Input</td>
<td>Definition file of the original Caffe model (.prototxt).</td>
<td>A string.</td>
</tr>
<tr>
<td>weights_file</td>
<td>Input</td>
<td>Weight file of the original Caffe model (.caffemodel).</td>
<td>A string.</td>
</tr>
<tr>
<td>config_definition</td>
<td>Input</td>
<td>Simplified quantization configuration file quant.cfg, which is generated from the retrain_config_caffe.proto file. The retrain_config_caffe.proto file is stored in /amct_caffe/proto/retrain_config_caffe.proto in the AMCT installation path. For details about the parameters in the retrain_config_caffe.proto template and the generated simplified quantization configuration file quant.cfg, see 3.11.3 Simplified Quantization Aware Training Configuration File.</td>
<td>Default: None A string. Restriction: If it is set to None, a configuration file is generated based on the residual arguments. Otherwise, a configuration file in JSON format is generated based on this argument.</td>
</tr>
</tbody>
</table>
Returns

None

Outputs

Outputs a quantization aware training configuration file in JSON format. The existing configuration file (if available) in the directory will be overwritten upon this API call. An example is as follows.

```json
{
  "version":1,
  "conv1":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo":"ulq_quantize"
    },
    "retrain_weight_config":{
      "algo":"arq_retrain",
      "channel_wise":true
    }
  },
  "conv2_1/expand":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo":"ulq_quantize"
    },
    "retrain_weight_config":{
      "algo":"arq_retrain",
      "channel_wise":true
    }
  },
  "conv2_1/dwise":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo":"ulq_quantize"
    },
    "retrain_weight_config":{
      "algo":"arq_retrain",
      "channel_wise":true
    }
  }
}
```

Example

```python
from amct.caffe import amct
retrain_simple = 'retrain/retrain.cfg'
model_file = 'resnet50_train.prototxt'
weights_file = 'ResNet-50-model.caffemodel'
config_json_file = './config.json'
amct.create_quant_retrain_config(config_json_file, model_file, weights_file, retrain_simple)
```

3.9.3.2 create_quant_retrain_model

Description

Applies to quantization aware training. Runs quantization aware training on a graph based on the config_file configuration file, inserts activations' and weights' fake-quantization layers, and saves the modified network to a new file.
Restrictions

None

Prototype

```python
create_quant_retrain_model(model_file, weights_file, config_file, modified_model_file, modified_weights_file, scale_offset_record_file)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input / Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_file</td>
<td>Input</td>
<td>Caffe model definition file (.prototxt).</td>
<td>A string.</td>
</tr>
<tr>
<td>weights_file</td>
<td>Input</td>
<td>Weight file of the already-trained Caffe model (.caffemodel).</td>
<td>A string.</td>
</tr>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization configuration file.</td>
<td>A string.</td>
</tr>
<tr>
<td>modified_model_file</td>
<td>Input</td>
<td>File name of the generated Caffe model definition file (.prototxt).</td>
<td>A string.</td>
</tr>
<tr>
<td>modified_weights_file</td>
<td>Input</td>
<td>File name of the generated Caffe model weight file (.caffemodel).</td>
<td>A string.</td>
</tr>
<tr>
<td>scale_offset_record_file</td>
<td>Input</td>
<td>Quantization factor record file.</td>
<td>A string.</td>
</tr>
</tbody>
</table>

Returns

None

Outputs

- Outputs `modified_model_file`, the definition file of the result model inserted with quantization aware training layers.
- Outputs `modified_weights_file`, the weight file of the result model inserted with quantization aware training layers.

Example

```python
from amct_caffe import amct
model_file = 'resnet50_train.prototxt'
weights_file = 'ResNet-50-model.caffemodel'
modified_model_file = './tmp/modified_model.prototxt'
modified_weights_file = './tmp/modified_model.caffemodel'
config_json_file = './config.json'
scale_offset_record_file = './record.txt'
amct.create_quant_retrain_model(model_file, weights_file, config_json_file, modified_model_file, modified_weights_file, scale_offset_record_file)
```
3.9.3.3 save_quant_retrain_model

Description

Applies to quantization aware training. Saves a retrained model to a model that can be used for both accuracy simulation and inference.

Restrictions

None

Prototype

```
save_quant_retrain_model(retrained_model_file, retrained_weights_file, save_type, save_path, scale_offset_record_file = None, config_file = None)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>retrained_model_file</td>
<td>Input</td>
<td>Generated Caffe model definition file (.prototxt).</td>
<td>A string. Restrictions: For layers for inference, the settings in LayerParameter in retrained_model_file must meet inference requirements. For example, use_global_stats of the BatchNorm layer must be set to 1.</td>
</tr>
<tr>
<td>retrained_weights_file</td>
<td>Input</td>
<td>Generated Caffe model weight file (.caffemodel).</td>
<td>A string.</td>
</tr>
<tr>
<td>save_type</td>
<td>Input</td>
<td>Type of the model to save:</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Fakequant: a model for accuracy simulation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Deploy: a model deployable on the Ascend AI Processor.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Both: two models, one for accuracy simulation and one deployable on the Ascend AI Processor.</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, <code>./quantized_model/*model</code>.</td>
<td>A string.</td>
</tr>
<tr>
<td>scale_offset_record_file</td>
<td>Input</td>
<td>Quantization factor record file.</td>
<td>A string. (Initialized to <strong>None</strong> by default.)</td>
</tr>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization configuration file.</td>
<td>A string. (Initialized to <strong>None</strong> by default.)</td>
</tr>
</tbody>
</table>

**Returns**

None

**Outputs**

- Outputs a model for accuracy simulation in the Caffe environment and its weight file, with names containing the **fake_quant** keyword.
- Outputs a deployable model file and its weight file, with names containing the **deploy** keyword. The model can be deployed on the Ascend AI Processor after being converted by the ATC tool.

**Example**

```python
from amctcaffe import amct
retrained_model_file = './pre_model/retrained_resnet50.prototxt'
retrained_weights_file = './pre_model/resnet50_solver_iter_35000.caffemodel'
scale_offset_record_file = './record.txt'
# Insert retrain layers, and save the quantized model to a .prototxt model file and a .caffemodel weight file. The following files can be found in the ./result directory: model_fake_quant_model.prototxt, model_fake_quant_weights.caffemodel, model_deploy_model.prototxt, and model_deploy_weights.prototxt.
amct.save_quant_retrain_model(retrained_model_file, retrained_weights_file, 'Both', './result/model', scale_offset_record_file, config_json_file)
```

### 3.9.4 Tensor Decomposition

#### 3.9.4.1 auto_decomposition

**Description**

Generates a decomposed model file and its weight file from a given original Caffe model (.prototxt) and its weight file (.caffemodel).
Restrictions

- Ensure that the input .prototxt file matches the input .caffemodel file.
- Pass the directory of the original model and an output directory to the call to the tensor decomposition API. The API automatically decomposes any Convolution layer that meets the decomposition conditions. For details about the decomposition conditions, see Restrictions.

Prototype

```python
auto_decomposition(model_file, weights_file, new_model_file, new_weights_file)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_file</td>
<td>Input</td>
<td>Caffe model definition file (.prototxt).</td>
<td>A string.</td>
</tr>
<tr>
<td>weights_file</td>
<td>Input</td>
<td>Weight file of the already-trained Caffe model (.caffemodel).</td>
<td>A string.</td>
</tr>
<tr>
<td>new_model_file</td>
<td>Input</td>
<td>Caffe model definition file (.prototxt) after tensor decomposition, for example, xx_tensor_decomposition.prototxt.</td>
<td>A string.</td>
</tr>
<tr>
<td>new_weights_file</td>
<td>Input</td>
<td>Caffe model weight file (.caffemodel) after tensor decomposition, for example, xx_tensor_decomposition.caffemodel.</td>
<td>A string.</td>
</tr>
</tbody>
</table>

Returns

None

Outputs

- Model definition file after tensor decomposition (.prototxt).
- Model weight file after tensor decomposition (.caffemodel).

Example

```python
from amct.caffe.tensor_decompose import auto_decomposition
auto_decomposition(model_file='ResNet-50-deploy.prototxt',
                    weights_file='ResNet-50-weights.caffemodel',
                    new_model_file='ResNet-50-deploy_tensor_decomposition.prototxt',
                    new_weights_file='ResNet-50-deploy_tensor_decomposition.caffemodel')
```
3.10 FAQs

3.10.1 Failed to Install the Caffe Environment in CLI Mode

Symptom

When the Caffe environment is installed in CLI mode, information similar to "/usr/bin/python3.7: can't open file '/usr/lib/python3.7/py_compile.py': [Error 2] No such file or directory" is displayed and the Caffe environment fails to be installed.

Possible Cause

Python 3.7.5 must be installed in advance before installing the AMCT. However, when Caffe 1.0 is installed in CLI mode, the system searches for `py_compile.py`. This file exists only in Python 3.6 or 2.7, but not in Python 3.7.5.

Solution

When installing Python 3.7.5, add the following soft link or copy the `py_compile.py` file in `/usr/local/python3.7.5/lib/python3.7` to `/usr/lib/python3.7`, and then install the Caffe environment in CLI mode.

Set the soft link:

```bash
sudo ln -s /usr/local/python3.7.5/lib/python3.7 /usr/lib/python3.7
```

If a message indicating that the links exist is displayed during the command execution, run the following commands to delete the existing links and run the preceding commands again:

```bash
sudo rm -rf /usr/lib/python3.7
```

3.10.2 An Error Message Is Displayed During python3-tk Installation

Symptom

When the python3-tk dependency is installed, the following error message is displayed.

```bash
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3-tk is already the newest version (3.6.9-1+deb9u4). The following packages were automatically installed and are no longer required:
  aptitude-common libboost-system1.58.0_1libc6ttc3v5 libc6tcs2v5 libjpeg4-2.0-0v5 libsocket2v5
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 394 not upgraded. I am not fully installed or removed.
```

```bash
Error: APT is trying to download 100% of
```

```bash
Sub-process /usr/bin/dpkg returned an error code (1)
```

Solution

Copy the missing file `py_compile.py` to the `/usr/lib/python3.7` directory and reinstall the Python.

```
cp /usr/local/python3.7.5/lib/python3.7/py_compile.py /usr/lib/python3.7
```

Replace `/usr/local/python3.7.5/lib/python3.7/py_compile.py` with the actual path of the file.

3.10.3 Proto Merging Error

1. The user message definition conflicts with the AMCT custom layer.

The content of `custom.proto` is as follows. The QuantParameter layer conflicts with that in `amct_custom.proto`.

```protobuf
message LayerParameter {
  optional QuantParameter quant_param = 208;
  optional ReLU6Parameter relu6_param = 1000000;
  optional ROIPoolingParameter roi_pooling_param = 8266711;
}
```

```protobuf
message ReLU6Parameter {
  optional float negative_slope = 1 [default = 0];
}
```

```protobuf
message ROIPoolingParameter {
  // Pad, kernel size, and stride are all given as a single value for equal
  // dimensions in height and width or as Y, X pairs.
  optional uint32 pooled_h = 1 [default = 0]; // The pooled output height
  optional uint32 pooled_w = 2 [default = 0]; // The pooled output width
  // Multiplicative spatial scale factor to translate ROI coords from their
  // input scale to the scale used when pooling
  optional float spatial_scale = 3 [default = 1];
}
```

```protobuf
message QuantParameter {
  optional bool with_offset = 1;
  optional float scale = 2;
  optional bytes offset = 3;
  optional string object_layer = 4;
}
```

The following error messages are displayed when the proto merging command is executed:

```
Merging failed, please check your custom.proto.
```

Solution 1

Modify the user message definition as prompted.
2. The custom layer index number defined in LayerParameter conflicts with that of the AMCT.

The content of `custom.proto` is as follows. The `LayerParameter` index number defined in `custom.proto` conflicts with that in `amct_custom.proto`.

```protobuf
message LayerParameter {
  optional QuantParameter quant_param = 208;
  optional ReLU6Parameter relu6_param = 1000000;
  optional ROIPoolingParameter roi_pooling_param = 8266711;
}

message ReLU6Parameter {
  optional float negative_slope = 1 [default = 0];
}

message ROIPoolingParameter {
  // Pad, kernel size, and stride are all given as a single value for equal
  // dimensions in height and width or as Y, X pairs.
  optional uint32 pooled_h = 1 [default = 0]; // The pooled output height
  optional uint32 pooled_w = 2 [default = 0]; // The pooled output width
  // Multiplicative spatial scale factor to translate ROI coords from their
  // input scale to the scale used when pooling
  optional float spatial_scale = 3 [default = 1];
}
```

The following error messages are displayed when the proto merging command is executed:

![Error Message](image)

**Solution 2**

Change the index number of the custom operator in `custom.proto` as prompted.

3. The custom layer index number defined in LayerParameter conflicts with that of the ATC.

The content of `custom.proto` is as follows. The `LayerParameter` index number defined in `custom.proto` conflicts with that in ATC `caffe.proto`.

```protobuf
message LayerParameter {
  optional ReLU6Parameter relu6_param = 206;
  optional ROIPoolingParameter roi_pooling_param = 8266711;
}

message ReLU6Parameter {
  optional float negative_slope = 1 [default = 0];
}

message ROIPoolingParameter {
  // Pad, kernel size, and stride are all given as a single value for equal
  // dimensions in height and width or as Y, X pairs.
  optional uint32 pooled_h = 1 [default = 0]; // The pooled output height
  optional uint32 pooled_w = 2 [default = 0]; // The pooled output width
  // Multiplicative spatial scale factor to translate ROI coords from their
  // input scale to the scale used when pooling
```
optional float spatial_scale = 3 [default = 1];
}

The following error messages are displayed when the proto merging command is executed.

```
Solution 3
Change the index number of the custom operator in `custom.proto` as prompted.

4. The user message definition conflicts with the ATC custom layer.

The content of `custom.proto` is as follows. The NormalizeParameter layer conflicts with that in `caffe.proto`.

```protobuf
message LayerParameter {
  optional ReLU6Parameter relu6_param = 1000000;
  optional ROIPoolingParameter roi_pooling_param = 8266711;
  optional NormalizeParameter norm_param = 206;
}
```
```protobuf
message ReLU6Parameter {
  optional float negative_slope = 1 [default = 0];
}
```
```protobuf
message ROIPoolingParameter {
  // Pad, kernel size, and stride are all given as a single value for equal
  // dimensions in height and width or as Y, X pairs.
  optional uint32 pooled_h = 1 [default = 0]; // The pooled output height
  optional uint32 pooled_w = 2 [default = 0]; // The pooled output width
  // Multiplicative spatial scale factor to translate ROI coords from their
  // input scale to the scale used when pooling
  optional float spatial_scale = 3 [default = 1];
}
```
```protobuf
message NormalizeParameter {
  optional bool across_spatial = 1 [default = true];
  // Initial value of scale. Default is 1.0 for all
  optional FillerParameter scale_filler = 2;
  // Whether or not scale parameters are shared across channels.
  optional bool channel_shared = 3 [default = true];
  // Epsilon for not dividing by zero while normalizing variance
  optional float eps = 4 [default = 1e-10];
}
```

No error message is displayed when the proto merging command is run. By default, the `custom.proto` file prevails and the ATC built-in message definition is overwritten. The following messages are displayed.

```
... Messages from the merge process...
```
Solution 4

None

3.10.4 "RuntimeError: Cannot find scale_d of layer '**' in record file" Is Displayed During Quantization

Symptom

When the `save_model` API is called to save the quantization model during quantization, the activation quantization parameters `scale_d` and `offset_d` computed in the calibration phase need to be read. If the corresponding parameters cannot be found in the corresponding record file, the quantization model cannot be saved. Therefore, the preceding AMCT error is reported and the process is terminated.

Possible Cause

The `scale_d` and `offset_d` parameters are saved during calibration (when the Caffe framework is called to perform forward calculation on the calibration model). The AMCT inserts the IFMR layer into the calibration model, and the IFMR layer needs to accumulate enough user-specified `batch_num` data and then perform quantization to obtain `scale_d` and `offset_d`. The causes of the "RuntimeError: Cannot find scale_d of layer * in record file" error are as follows:

1. An error occurs when the Caffe inference is executed. The possible causes are as follows: The compiled Caffe is faulty, the calibration model is faulty, or the corresponding dataset cannot be found. You can view the exception information thrown by the Caffe framework.

2. The data amount of the calibration dataset provided by the user does not meet the data amount required by `batch_num`. For example, if the user provides only one batch of data as the calibration dataset and `batch_num` is set to 2, the IFMR layer does not store sufficient data during calibration, the quantization operation cannot be performed. As a result, `scale_d` and `offset_d` cannot be computed, and the preceding error is also triggered. You can view the process information printed during the quantization of the IFMR to locate the fault. The amount of saved data is displayed in the IFMR layer.
Solution

1. Rectify the fault based on the error reported by the Caffe framework.
2. Increase the data volume of the calibration dataset or decrease the value of `batch_num` for the quantization algorithm until the data volume of the calibration dataset is greater than or equal to the value of `batch_num`. Note that decreasing the value of `batch_num` may reduce the model accuracy after quantization.

3.10.5 "UserWarning: Matplotlib is currently using agg, which is a non-GUI backend, so cannot show the figure." Is Displayed When Quantizing an Object Detection Network

Symptom

When an object detection network is quantized, the "UserWarning: Matplotlib is currently using agg, which is a non-GUI backend, so cannot show the figure." message is displayed. As a result, the quantized detection result is not displayed on the GUI.

Possible Cause

The possible cause is that the Tkinter is not installed. You can run the following command in the Python terminal to verify the installation. If the following information is displayed, the Tkinter is not installed successfully.

```bash
hisisoc@ubuntu62:$ python3.7.5
Python 3.7.5 (default, Mar  3 2020, 13:58:02)
[GCC 7.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tkinter
Traceback (most recent call last):
  File "<stdin>"., line 1, in <module>
  File /usr/local/python3.7.5/lib/python3.7/tkinter/__init__.py", line 36, in <module>
      import _tkinter # If this fails your Python may not be configured for Tk
ModuleNotFoundError: No module named '_tkinter'
```

Solution

If the Tkinter fails to be installed by installing python3-tk, the possible cause is that multiple Python 3 versions are installed or the tk-dev library fails to be installed. If the tk-dev library fails to be installed, perform the following steps:

1. Run the following command to reinstall the tk-dev library:
   ```bash
   sudo apt-get install tk-dev
   ```
2. Go to the Python 3.7.5 installation directory and recompile and install Python 3.7.5.
   ```bash
   cd Python-3.7.5
   ./configure --prefix=/usr/local/python3.7.5 --enable-shared
   make
   sudo make install
   ```
3. Delete the original soft links.
   ```bash
   sudo rm -rf /usr/bin/python3.7.5
   sudo rm -rf /usr/bin/pip3.7.5
   sudo rm -rf /usr/bin/python3.7
   sudo rm -rf /usr/bin/pip3.7
   ```
4. Set the soft links again.
   ```
   sudo ln -s /usr/local/python3.7.5/bin/python3 /usr/bin/python3.7.5
   sudo ln -s /usr/local/python3.7.5/bin/pip3 /usr/bin/pip3.7.5
   sudo ln -s /usr/local/python3.7.5/bin/python3 /usr/bin/python3.7
   sudo ln -s /usr/local/python3.7.5/bin/pip3 /usr/bin/pip3.7
   ```

5. Verify the installation again. If `import tkinter` is displayed, the installation is successful.
   ```
   hisisoc@ubuntu62:~$ python3.7.5
   Python 3.7.5 (default, Mar 3 2020, 13:58:02)
   [GCC 7.4.0] on linux
   Type "help", "copyright", "credits" or "license" for more information.
   >>> import tkinter
   >>>
   ```

### 3.10.6 What Do I Do If "IfmrQuantWithOffset scale is illegal" Is Displayed During Calibration?

#### Symptom

When the Caffe framework is used to perform intermediate calibration model inference, the scale computed by the quantization algorithm is improper due to the invalid input data range. As a result, the calibration fails and the Caffe calibration process is terminated.

#### Possible Cause

1. Data range \([-\infty, +\infty]\):
   
   The quantization algorithm of AMCT requires zero crossing. Therefore, the computed scale is \(\text{inf}/255 = \text{inf}\). In this case, the quantization factor cannot be carried. Therefore, an error message is displayed for the quantization algorithm indicating that the data range is not required if the scale is \(\text{inf}\) after quantization, the system displays a message indicating that the data range is not supported and displays an error message.

   ![Figure 3-9 Error message 1](image)

2. Data range: \([\text{DBL_EPSILON}, \text{FLT_EPSILON}]\) (\(\text{EPSILON}\) includes the \(\text{DBL_EPSILON}\) double type and \(\text{FLT_EPSILON}\) float type. Currently, the \(\text{FLT_EPSILON}\) type is used.)

   The AMCT quantization supports the maximum value
   
   \[
   \text{scale} = \frac{1}{\text{FLT\_EPSILON}}
   \]

   obtained through calculation, because the the Ascend AI Processor quantization uses multiplication calculation:

   \[
   \text{int8} = \text{round}\left(\text{float} \times \frac{1}{\text{scale}}\right)
   \]

   If scale is greater than \(\frac{1}{\text{FLT\_EPSILON}}\),
1. scale is less than FLT_EPSILON. In this case, the quantization result is unreliable. Therefore, the AMCT quantization algorithm supports original activation quantization only within the range of

\[
\left[ -\frac{128}{\text{FLT\_EPSILON}}, \frac{127}{\text{FLT\_EPSILON}} \right]
\]

Otherwise, an error message is displayed, indicating that the quantization algorithm is not supported.

Figure 3-10 Error message 2

3.11 Appendixes

3.11.1 Proto Merging Principles

Figure 3-11 shows the merging principles.

- AMCT is short for the Ascend Model Compression Toolkit.
- ATC is short for the Ascend Tensor Compiler.

The .proto files are described as follows:

- **custom.proto**: custom .proto file.
- **amct_custom.proto**: AMCT .proto file, including the AMCT custom layers and updated layers of Caffe-master compared with Caffe 1.0.
- **caffe.proto**: ATC .proto file, including the ATC custom layers and updated layers compared with Caffe 1.0, with layer sequence numbers updated. This file is incorporated into the AMCT tar.gz package.

Figure 3-11 Proto merging principles
The merging workflow and principles are described as follows:

1. Prepare the `custom.proto` file and run the `install.py` script provided by AMCT. The script merges the `custom.proto` file with the `amct_custom.proto` file to generate a temporary `custom.proto` file.
   - If `custom.proto` and `amct_custom.proto` have the same operator sequence numbers, an error is reported, prompting the user to change the operator sequence number in `custom.proto`.
   - If `custom.proto` and `amct_custom.proto` have the same operator names, an error is reported, prompting the user to change the operator name in `custom.proto`.

2. Merge the generated temporary `custom.proto` file with `caffe.proto` in the ATC runfile to generate the final `caffe.proto` file.
   - If `custom.proto` and `caffe.proto` have the same operator sequence numbers, an error is reported, prompting the user to change the operator sequence number in `custom.proto`.
   - If `custom.proto` and `caffe.proto` have the same operator names, the duplicate operator in `custom.proto` is retained while the duplicate operator in `caffe.proto` is removed.

3. Find the `caffe.proto` file in the Caffe project according to the `caffe_dir` directory specified by the user, and back up and replace it.

### NOTE

- The layers in `amct_custom.proto` are numbered starting at 200000.
- The sequence numbers of the ATC custom layers in `caffe.proto` are within the range [5000, 200000].
- Keep the sequence number a custom layer in `custom.proto` within 5000 and avoid conflict with the built-in layers in the ATC `caffe.proto` file.

### 3.11.2 Simplified Post-training Quantization Configuration File

Table 3-43 describes the parameters in the `calibration_config_caffe.proto` template.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified post-training quantization configuration file of AMCT.</td>
</tr>
<tr>
<td>Optimal</td>
<td>Optional</td>
<td>uint32</td>
<td>batch_num</td>
<td>Batch count used for quantization.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
<td>--------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>activation_offset</td>
<td>Whether to quantize activations with offset.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>joint_quant</td>
<td>Eltwise joint quantization switch. Defaults to false, indicating that joint quantization is disabled. If it is set to true, the network performance may improve but the accuracy may be compromised.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_layers</td>
<td>Layers to skip during quantization.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_layer_types</td>
<td>Types of layers to skip during quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>Nuq Config</td>
<td>nuq_config</td>
<td>Non-uniform quantization configuration.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>CalibrationConfig</td>
<td>common_config</td>
<td>Common quantization configuration. If a layer is not overridden by override_layer_types or override_layer_configs, this configuration is used.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>OverrideLayerType</td>
<td>override_layer_types</td>
<td>Type of layers to override.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>OverrideLayer</td>
<td>override_layer_confs</td>
<td>Layer to override.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>do_fusion</td>
<td>BN fusion switch. Defaults to true.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_fusion_layers</td>
<td>Layers to skip in BN fusion.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------</td>
<td>----------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Optinal</td>
<td>Calib</td>
<td>conv_calibration_config</td>
<td>Quantization configuration of the Convolution and Deconvolution layers that are not overridden. Deprecated, which is not recommended.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>Calib</td>
<td>fc_calibration_config</td>
<td>Quantization configuration of InnerProduct and AVE Pooling layers that are not overridden. Deprecated, which is not recommended.</td>
</tr>
<tr>
<td>NuqConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Non-uniform quantization configuration.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>String</td>
<td>mapping_file</td>
<td>JSON file converted from the offline model converted using the ATC tool from the deployable model generated form uniform quantization.</td>
</tr>
<tr>
<td></td>
<td>Optimal</td>
<td>NU</td>
<td>nuq_quantize</td>
<td>Non-uniform quantization configuration.</td>
</tr>
<tr>
<td>OverrideLayerType</td>
<td>Required</td>
<td>String</td>
<td>layer_type</td>
<td>Type of layers to quantize.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>Calib</td>
<td>calibration_config</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td>OverrideLayer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization configuration overriding by layer.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>String</td>
<td>layer_name</td>
<td>Layer to override.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
<td>-----------------------</td>
<td>-------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>CalibrationConfig</td>
<td>Required</td>
<td>CalibrationConfig</td>
<td>calibration_config</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>calibrate</td>
<td>Calibration-based quantization configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>arq_quantize</td>
<td>Weight quantization algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>nuq_quantize</td>
<td>Weight quantization algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ifmr_quantize</td>
<td>Activation quantization algorithm.</td>
</tr>
<tr>
<td>ARQuantize</td>
<td>Optional</td>
<td>bool</td>
<td>channel_wise</td>
<td>Whether to use different quantization factors for each channel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>search_range_start</td>
<td>Quantization factor search start.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>search_range_end</td>
<td>Quantization factor search end.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>search_step</td>
<td>Quantization factor search step.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>max_percentile</td>
<td>Upper search limit.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
<td>-------</td>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>min_percentile</td>
<td>Lower search limit.</td>
</tr>
<tr>
<td>NUQuantize</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Non-uniform quantization configuration.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>uint 32</td>
<td>num_steps</td>
<td>Number of steps for non-uniform quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>uint 32</td>
<td>num_of_iteration</td>
<td>Number of iterations for non-uniform quantization optimization.</td>
</tr>
</tbody>
</table>

The following is an example simplified uniform quantization configuration file (quant.cfg).

```
# Global quantization parameters
batch_num : 2
activation_offset : true
joint_quant : false
skip_layers : "conv_1"
skip_layer_types:"Convolution"
do_fusion: true
skip_fusion_layers : "conv_1"
common_config : {
    arq_quantize : {
        channel_wise : true
    }
    ifmr_quantize : {
        search_range_start : 0.7
        search_range_end : 1.3
        search_step : 0.01
        max_percentile : 0.999999
        min_percentile : 0.999999
    }
}
override_layer_types : {
    layer_type : "InnerProduct"
calibration_config : {
        arq_quantize : {
            channel_wise : false
        }
        ifmr_quantize : {
            search_range_start : 0.8
            search_range_end : 1.2
            search_step : 0.02
            max_percentile : 0.999999
            min_percentile : 0.999999
        }
    }
}
override_layer_configs : {
```

layer_name : "conv_2"
calibration_config : {
  arq_quantize : {
    channel_wise : true
  }
  ifmr_quantize : {
    search_range_start : 0.8
    search_range_end : 1.2
    search_step : 0.02
    max_percentile : 0.999999
    min_percentile : 0.999999
  }
}

3.11.3 Simplified Quantization Aware Training Configuration File

Table 3-44 describes the parameters in the `retrain_config_caffe.proto` template.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTRetrainConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified quantization aware training configuration file of AMCT.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>RetrainDataQuantConfig</td>
<td>retrain_data_quant_config</td>
<td>Activation retrain configuration.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
<td>------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>RetrainWeightQuantConfig</td>
<td>Required</td>
<td>Retrain</td>
<td>retrain_weight_quant_config</td>
<td>Weight retrain configuration.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>String</td>
<td>skip_layers</td>
<td>Layers to skip.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>Retrain</td>
<td>override_layer_configs</td>
<td>Layer to override.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>String</td>
<td>skip_layer_types</td>
<td>Types of layers to skip. This parameter is not supported in the current version.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>Retrain</td>
<td>override_layer_types</td>
<td>Types of layers to override.</td>
</tr>
<tr>
<td>RetrainDataQuantConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Activation retrain configuration.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------</td>
<td>--------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ULQuantize</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ULQ algorithm configuration.</td>
</tr>
<tr>
<td>ClipMaxMin</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Initial upper and lower limits.</td>
</tr>
<tr>
<td>RetrainWeightQuantConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Weight retrain configuration.</td>
</tr>
<tr>
<td>ARQRetrain</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ARQ algorithm configuration.</td>
</tr>
<tr>
<td>RetrainOverrideLayer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Layer overriding configuration.</td>
</tr>
</tbody>
</table>

- **ULQ** quantize
- **clip_max_min**
- **clip_max**
- **clip_min**
- **arq_retrain**
- **channel_wise**
<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>layer_name</td>
<td>Layer name.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>retrain_data_quant_config</td>
<td>Activation quantization configuration to apply.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>retrain_weight_quant_config</td>
<td>Weight quantization configuration to apply.</td>
</tr>
<tr>
<td>RetrainOverrideLayerType</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Types of layers to override.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>layer_type</td>
<td>Layer type.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>retrain_data_quant_config</td>
<td>Activation quantization configuration to apply.</td>
</tr>
</tbody>
</table>
The following is an example simplified quantization aware training configuration file (*quant.cfg*).

```python
# Global quantization parameters
retrain_data_quant_config: {
    ulq_quantize: {
        clip_max_min: {
            clip_max: 6.0
            clip_min: -6.0
        }
    }
}

retrain_weight_quant_config: {
    arq_retrain: {
        channel_wise: true
    }
}

skip_layers: "conv_1"

override_layer_types : {
    layer_type: "InnerProduct"
    retrain_weight_quant_config: {
        arg_retrain: {
            channel_wise: false
        }
    }
}

override_layer_configs : {
    layer_name: "fc_5"
    retrain_weight_quant_config: {
        arg_retrain: {
            channel_wise: false
        }
    }
}
```
3.11.4 Quantization Factor Record File

Prototype

The quantization factor record file is a serialized data structure file based on Protobuf. You can generate a quantized model file by using the quantization configuration file, original network model file, and the quantization factor record file. The Protobuf prototype is defined as follows.

```protobuf
message SingleLayerRecord {
  optional float scale_d = 1;
  optional int32 offset_d = 2;
  repeated float scale_w = 3;
  repeated int32 offset_w = 4;
  repeated uint32 shift_bit = 5;
  optional uint32 channels = 6;
  optional uint32 height = 7;
  optional uint32 width = 8;
  optional bool skip_fusion = 9 [default = false];
}
message ScaleOffsetRecord {
  message MapFiledEntry {
    optional string key = 1;
    optional SingleLayerRecord value = 2;
  } repeated MapFiledEntry record = 1;
}
```

The parameters are described as follows.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScaleOffsetRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Map structure. To ensure compatibility, the discrete map structure is used.</td>
</tr>
<tr>
<td>Repeated</td>
<td></td>
<td>MapFiledEntry</td>
<td>record</td>
<td>Each records a quantization factor of a quantization layer and consists of two members:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• <strong>key</strong>: layer name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• <strong>value</strong>: quantization factor defined by SingleLayerRecord.</td>
</tr>
<tr>
<td>SingleLayerRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization factors for quantization.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>float</td>
<td>scale_d</td>
<td>Scale factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>int32</td>
<td>offset_d</td>
<td>Offset factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>---------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Repeated</td>
<td>float</td>
<td>scale_w</td>
<td>Scale factor for weight quantization. Scalar mode (quantizing the weight of the current layer in a unified manner) and vector mode (quantizing the weight of the current layer in channel-wise mode) are supported. Only Convolution and Deconvolution layers support channel-wise quantization.</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>int32</td>
<td>offset_w</td>
<td>Offset factor for weight quantization. Similar to scale_w, it also supports the scalar and vector quantization modes and the dimension configuration must be the same as that of scale_w. Currently weight quantization with offset is not supported, and offset_w must be 0.</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>uint32</td>
<td>shift_bit</td>
<td>Shift factor. Reserved for the 3.9.2.5 convert_model API.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>uint32</td>
<td>channels</td>
<td>Network-wide infer_shape is not supported. Therefore, the input shape of the current layer needs to be configured. This field is used to configure the size of the input channel dimension.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>uint32</td>
<td>height</td>
<td>Network-wide infer_shape is not supported. Therefore, the input shape of the current layer needs to be configured. This field is used to configure the size of the input height dimension.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>uint32</td>
<td>width</td>
<td>Network-wide infer_shape is not supported. Therefore, the input shape of the current layer needs to be configured. This field is used to configure the size of the input width dimension.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>skip_fusion</td>
<td>Whether to skip Conv+BN+Scale fusion, Deconv+BN+Scale fusion, BN+Scale +Conv fusion, and FC+BN+Scale fusion at the current layer. Defaults to false, indicating that fusion of the preceding types is performed.</td>
<td></td>
</tr>
</tbody>
</table>

Beware that the Protobuf protocol does not report an error for repeated settings of optional fields. Instead, the most recent settings are used.

For general quantization layers, a range of parameters need to be configured, including scale\_d, offset\_d, scale\_w, offset\_w, channels, height, width, and...
**shift_bit.** The scale_w and offset_w parameters are unavailable for AVE Pooling since the layer has no weight. An example of the quantization factor record file is as follows:

```plaintext
record {
  key: "conv1"
  value: {
    scale_d: 0.01424
    offset_d: -128
    scale_w: 0.43213
    scale_w: 0.78163
    scale_w: 1.03213
    offset_w: 0
    offset_w: 0
    offset_w: 0
    shift_bit: 1
    shift_bit: 1
    shift_bit: 1
    channels: 3
    height: 144
    width: 144
    skip_fusion: true
  }
}
record {
  key: "pool1"
  value: {
    scale_d: 0.532532
    offset_d: 13
    channels: 256
    height: 32
    width: 32
  }
}
record {
  key: "fc1"
  value: {
    scale_d: 0.37532
    offset_d: -67
    scale_w: 0.876221
    offset_w: 0
    shift_bit: 1
    channels: 1024
    height: 1
    width: 1
  }
}
```

**Quantization Factors**

The scale (a floating-point number) and offset quantization factors need to be provided for data and weight quantization. AMCT uses a unified quantization data structure. See the following expression.

\[
data_{\text{int8}} = \text{clip}_{\text{int8}}\left(\text{round}\left(\frac{data_{\text{float}}}{\text{scale}}\right) + \text{offset}\right)
\]

The value ranges are as follows:

- \(\text{scale} \in \left[\frac{1}{\text{FLT\_EPSILON}}, \text{FLT\_EPSILON}\right]\), where \(\text{FLT\_EPSILON} \approx 1.1920929 \times 10^{-7}\)
- \(\text{offset} \in [-128, 127]\)
Quantization algorithms are classified into symmetric quantization algorithms and asymmetric quantization algorithms.

1. Symmetric quantization algorithm:
The original high-precision data and quantized int8 data are converted into $\text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int8}}$, where $\text{scale}$ is a float32. To indicate positive and negative numbers, the signed int8 data type is used for $\text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int8}}$. The following describes how to convert the original data into the int8 format. $\text{round}$ is a rounding function. The value to be determined by the quantization algorithm is the constant $\text{scale}$.

$\text{data}_{\text{int8}} = \text{round}\left(\frac{\text{data}_{\text{float}}}{\text{scale}}\right)$

Quantization of the weights and activations may be summarized as a process of searching for a scale. Because $\text{data}_{\text{int8}}$ is a signed number, to ensure symmetry of the ranges represented by positive and negative values, an absolute value operation is first performed on all data, so that the range of the to-be-quantized data is changed to $\text{data}_{\text{int8}}$, and then $\text{scale}$ is determined. The range of positive int8 values is $[0, 127]$. Therefore, $\text{scale}$ can be computed as follows:

$\text{scale} = \frac{\text{data}_{\text{max}}}{127}$

Therefore, the range of the int8 values is $[-128 \times \text{scale}, 127 \times \text{scale}]$. Data beyond the range $[-128 \times \text{scale}, 127 \times \text{scale}]$ is saturated to a boundary value, and then the quantization operation shown in the formula is performed.

2. Asymmetric quantization algorithm:
The difference from symmetric quantization algorithms lies in the data conversion mode. The $\text{scale}$ and $\text{offset}$ constants also need to be determined. $\text{data}_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset})$

The uint8 data is converted through calculation based on the original high-accuracy data, which is shown in the following formula:

$\text{data}_{\text{uint8}} = \text{round}\left(\frac{\text{data}_{\text{float}}}{\text{scale}} - \text{offset}\right)$

$\text{scale}$ is an fp32, $\text{data}_{\text{uint8}}$ is an unsigned int8, and $\text{offset}$ is an int8. The data range is $[\text{scale} \times \text{offset}, \text{scale} \times (255 + \text{offset})]$. If a value range of the to-be-quantized data is $[\text{data}_{\text{min}}, \text{data}_{\text{max}}]$, $\text{scale}$ and $\text{offset}$ are computed as follows:

$\text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}$, \hspace{1cm} \text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}$

Unified quantization data format: By performing simple data conversion of the asymmetric quantization algorithm formula, the quantized data and the symmetric quantization algorithm are of the same type, that is, int. The specific conversion process is as follows:

int8 quantization is used as an example for description. The input original floating-point data is $\text{data}_{\text{float}}$, the original quantized fixed-point number is...
\[ \text{data}_{\text{float}} \text{, the quantization scale is scale, and the quantization offset is} \] \[ \text{data}_{\text{float}} \text{ (the algorithm requires zero crossing to avoid too much accuracy drop). The calculation principle of quantization is as follows:} \]
\[ \text{data}_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset}) = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset + 128}) = \text{scale} \times (\text{data}_{\text{int8}} - \text{offset}) \]

Where,
\[ \text{data}_{\text{int8}} = \text{data}_{\text{uint8}} - 128 \in [-128, 127], \text{offset} = -(\text{offset} + 128) \in [-128, 127] \]

Through the foregoing conversion, the data may also be converted into the int8 format. After scale and the converted offset are determined, the int8 data converted from the original floating-point data is as follows:
\[ \text{data}_{\text{int8}} = \text{clip} \left( \text{round} \left( \frac{\text{data}_{\text{float}}}{\text{scale}} \right) + \text{offset'} \right) \]

### 3.11.5 Python 3.7.5 Installation on Ubuntu

**Step 1** Check that the Python 3.7.5 development environment is available.

Run the `python3.7.5 --version`, `python3.7 --version`, `pip3.7.5 --version`, and `pip3.7 --version` commands to check whether the environment is available. If the following information is displayed, the environment is available. Otherwise, go to the next step.

```
Python 3.7.5
pip 19.2.3 from /usr/local/python3.7.5/lib/python3.7/site-packages/pip (python 3.7)
```

**Step 2** Install the Python 3.7.5 dependencies.

```
sudo apt-get install -y make zlib1g zlib1g-dev build-essential libbz2-dev libsqlite3-dev libssl-dev libxml2-dev libffi-dev openssl python3-tk
```

libsqlite3-dev must be installed before the Python installation. If the Python 3.7.5 environment has been installed in the user's OS before the libsqlite3-dev installation, you need to rebuild the Python environment. If python3-tk fails to be installed, see 3.10.2 An Error Message Is Displayed During python3-tk Installation.

**Step 3** Install Python 3.7.5.

1. Run the `wget` command to download the source package of Python 3.7.5 to any directory on the server where AMCT is located:
   `wget https://www.python.org/ftp/python/3.7.5/Python-3.7.5.tgz`

2. Go to the download directory and decompress the source package:
   `tar -zxvf Python-3.7.5.tgz`

3. Go to the new folder and run the following configuration, build, and installation commands:
   `cd Python-3.7.5`
   `/configure --prefix=/usr/local/python3.7.5 --enable-loadable-sqlite-extensions --enable-shared`
   `make`
   `sudo make install`

   --prefix specifies the Python installation path. You can modify it as required.
   --enable-shared is used to build the libpython3.7m.so.1.0 dynamic library. --enable-loadable-sqlite-extensions is used to load the sqlite-devel dependency.

This document uses --prefix=/usr/local/python3.7.5 as an example. After the configuration, build, and installation commands are executed, the installation package is output to the /usr/local/python3.7.5 directory, and the
libpython3.7m.so.1.0 dynamic library is output to the /usr/local/python3.7.5/lib/libpython3.7m.so.1.0 directory.

4. Set the soft links:
   
   ```
   sudo ln -s /usr/local/python3.7.5/bin/python3 /usr/local/python3.7.5/bin/python3.7.5
   sudo ln -s /usr/local/python3.7.5/bin/pip3 /usr/local/python3.7.5/bin/pip3.7.5
   ```

5. Set the Python 3.7.5 environment variables.
   
   a. If Python is installed by the root user:
      
      AMCT is installed by the root user. Run the following commands in the current terminal window to set environment variables:
      
      ```
      # Set the Python 3.7.5 library path.
      export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
      # If multiple Python 3 versions exist in the user environment, specify Python 3.7.5.
      export PATH=/usr/local/python3.7.5/bin:$PATH
      ```
      
      **NOTICE**
      
      If the running user is the root user, it is not advised to modify the .bashrc file. Otherwise, the Python tools provided by other systems may be unavailable. If you want to use the default tool, open another terminal window.

   b. If Python is installed by a non-root user:
      
      AMCT is also installed by the non-root user. Run the `vi ~/.bashrc` command in any directory to open the .bashrc file, and append the following lines to the file:
      
      ```
      # Set the Python 3.7.5 library path.
      export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
      # If multiple Python 3 versions exist in the user environment, specify Python 3.7.5.
      export PATH=/usr/local/python3.7.5/bin:$PATH
      ```
      
      Run the `:wq!` command to save the file and exit. Run the `source ~/.bashrc` command for the modification to take effect immediately.

**Step 4** After the installation is complete, run the following commands to check the installed version. If the required version information is displayed, the installation is successful.

```
python3.7.5 --version
pip3.7.5 --version
python3.7 --version
pip3.7 --version
```
4 AMCT Instructions (PyTorch)

4.1 Introduction

4.1.1 Overview

This document describes how to quantize a PyTorch model using Ascend Model Compression Toolkit (AMCT). In the quantization process, the precision of model weights and activations is reduced to make the model more compact, improving the compute efficiency while lowering the transfer latency.

AMCT is a PyTorch-based Python toolkit that implements Conv+BN fusion (Conv+BN fusion is performed on the "torch.nn.Conv2d+torch.nn.BatchNorm2d" composite in the model before AMCT quantization) as well as 8-bit quantization of activations and weights in neural networks. This toolkit decouples model quantization from model conversion. It implements independent quantization of quantization-capable operators in a model, and outputs an .onnx model file. The obtained accuracy simulation model can run on CPU or GPU to complete accuracy simulation. The obtained deployable model can run on the Ascend AI Processor with improved inference performance. This tool has the following advantages:

- Lightweight: You only need to install the tool.
• Easy-to-use APIs: You can complete quantization using APIs based on the PyTorch inference script.

• Hardware compatibility: The generated deployable model can be converted to an offline model by using the ATC tool to implement 8-bit inference on Ascend AI Processor.

• Configurable quantization: You can modify the quantization configuration file and adjust the quantization strategy to obtain the optimal quantization result.

Figure 4-1 shows the application scenarios. Before you run inference on the Ascend AI Processor with the model quantized by this tool, you need to use Ascend Tensor Compiler (ATC) to convert the quantized model to an offline model adapted to the Ascend AI Processor.

4.1.2 Features

4.1.2.1 Terminology

There are two forms of quantization: post-training quantization and quantization aware training. The foregoing two quantization forms are classified into weight quantization and activation quantization according to the quantization object.

As used in this document, the following terms have the meanings specified below.

Post-training Quantization

Post-training quantization refers to quantizing the weights of an already-trained model from float32 to int8 and calibrating and quantizing the activations at inference time by using a small calibration dataset. For details about the quantization workflow, see 4.3 Post-training Quantization.

• Calibration dataset

During the calibration process, the algorithm uses each piece of data in the calibration dataset as input, accumulates the input data of each layer (or operator) that needs to be quantized, and uses the accumulated data as input data of the quantization algorithm to determine the quantization factors. Because the quantization factors and the accuracy of the quantized model are closely related to the selection of the calibration dataset, you are advised to calibrate the model using a subset of images from the validation dataset.
### Activation quantization

Activation quantization is to collect statistics on the input data of each layer (or operator) to be quantized, to find the optimal pair of scale and offset per layer (or operator). For details, see [4.9.3 Quantization Factor Record File](#).

Activations are the intermediate results of model inference computation. The value ranges are input-specific. Therefore, a group of reference inputs (a calibration dataset) needs to be used as incentives to record the input data of each layer (or operator) to be quantized for searching for quantization factors (scale and offset). During data calibration, extra memory (video memory/RAM) is needed to store the input data used to determine the quantization factors. Therefore, the video memory/RAM usage is higher than that required for performing inference only. The size of the extra memory is positively correlated with `batch_size x batch_num` during calibration.

### Weight quantization

After model training, the weights and the value ranges are determined. Therefore, quantization may be directly performed based on the value range of each weight.

The layers that support post-training quantization are listed as follows.

#### Table 4-1 Layers that support post-training quantization and restrictions

<table>
<thead>
<tr>
<th>Layer</th>
<th>Restrictions</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>torch.nn.Linear</td>
<td>-</td>
<td>Layers sharing the weight and bias parameters do not support quantization.</td>
</tr>
<tr>
<td>torch.nn.Conv2d</td>
<td>padding_mode = zeros</td>
<td></td>
</tr>
<tr>
<td>torch.nn.ConvTranspose2d</td>
<td>dilation = 1, groups = 1, padding_mode = zeros</td>
<td></td>
</tr>
</tbody>
</table>

### Quantization Aware Training

Quantization aware training introduces quantization in the forward pass of the training process, allowing for higher accuracy.

Quantization aware training is time consuming and data hungry. For details about the quantization workflow, see [4.4 Quantization Aware Training](#).

- **Training dataset**
  - Dataset of the already-trained network.

- **Activation quantization**
  - Activation quantization refers to iterative training of the upper clip limit and lower clip limit, which are used to calculate the current scale and offset. The activation is the intermediate result of model inference and calculation. The ULQ retrain algorithm is used to continuously optimize the two factors during the quantization aware training process to obtain the optimal factors.

- **Weight quantization**
  - Weight quantization means to optimize the quantization parameters of weights during the quantization aware training process to obtain the optimal parameters.
The layers that support quantization aware training are listed as follows.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Restrictions</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>torch.nn.Linear</td>
<td>-</td>
<td>Layers sharing the weight and bias parameters do not support quantization.</td>
</tr>
<tr>
<td>torch.nn.Conv2d</td>
<td>padding_mode = zeros</td>
<td></td>
</tr>
</tbody>
</table>

### 4.1.2.2 Principles

**Figure 4-2** shows the AMCT principles. The blue parts are implemented by the user, and the gray part is implemented by the AMCT APIs called by the user.

- **Post-training Quantization**
  
  a. Construct an original PyTorch model and then call the [create_quant_config](#) API to generate a quantization configuration file.
  
  b. Call the [quantize_model](#) API to optimize the original PyTorch model based on the quantization configuration file. The optimized model contains operators of the quantization algorithm. Then, perform forward inference with the optimized model in the PyTorch environment based on the downloaded calibration dataset to obtain the quantization factors.
  
  c. Call the [save_model](#) API to save the quantized model to a model for accuracy simulation in the ONNX Runtime environment and a model deployable on the the Ascend AI Processor.

- **Quantization Aware Training**
  
  a. Construct an original PyTorch model and then call the [create_quant_retrain_config](#) API to generate a quantization configuration file.
  
  b. Call the [create_quant_retrain_model](#) API to modify the original model. The modified model contains operators of the quantization aware training algorithm.
  
  c. Train the modified model. If the training process is not interrupted, perform inference on the trained model. During the inference process, the quantization factors are written into the quantization factor record file. Then, call the [save_quant_retrain_model](#) API to save a model for accuracy simulation and a deployable model. If the training process is interrupted, call the [restore_quant_retrain_model](#) API again based on the saved .pth model parameters and quantization configuration file to output a modified retrained network for further quantization-aware training. Then, perform inference on the result model, and call the [save_quant_retrain_model](#) API to save the quantized model.

For details about APIs, see [4.7 APIs](#).
4.1.3 Tool Workflow

**Figure 4-3** shows the tool workflow.

**Figure 4-3** Tool Workflow

- **Package preparation**
- **Pre-installation actions**
- **Installation**
  - Script creation with AMCT API calls
  - Quantization
    - Quantized deployable model (.onn)
- **Model conversion using ATC**
  - Model file (.om)
  - Ascend AI Processor

- **1. Create a user**
- **2. Check the system requirements and environment**
- **3. Install dependencies**
- **4. Upload the package**

- **Prepare a pre-trained model**
- **Prepare a dataset**
<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package preparation</td>
<td>Obtain the tool package by referring to <a href="#">4.2.1 Package Preparation</a>.</td>
</tr>
<tr>
<td>Pre-installation actions</td>
<td>Before AMCT installation, create an installation user, check the system environment, install dependencies, and upload the AMCT package. For details, see <a href="#">4.2.2 Pre-installation Actions</a>.</td>
</tr>
<tr>
<td>Installation</td>
<td>Install the PyTorch AMCT by referring to <a href="#">4.2.3 Installation</a>.</td>
</tr>
<tr>
<td>(Optional) Script creation with AMCT API calls</td>
<td>If you need to quantize your network model instead of the sample model provided in this instruction, you need to modify the script for adaptation before quantization.</td>
</tr>
<tr>
<td>Quantization</td>
<td>There are two forms of quantization: <a href="#">4.3 Post-training Quantization</a> and <a href="#">4.4 Quantization Aware Training</a>.</td>
</tr>
<tr>
<td>(Optional) Model conversion using ATC</td>
<td>You can convert the quantized deployable model to an offline model supported by the Ascend AI Processor by using ATC, and then perform subsequent inference.</td>
</tr>
</tbody>
</table>

### 4.2 AMCT Installation

#### 4.2.1 Package Preparation

Currently, AMCT runs only on Ubuntu 18.04 (x86_64) servers. Before installation, click [here](#) to obtain the AMCT package Ascend-cann-amct_{software version}_ubuntu18.04-x86_64.tar.gz.

Before installation, obtain the AMCT package. AMCT runs on Ubuntu 18.04 (x86_64) or EulerOS (AArch64) servers. Select a required software package.

- Ubuntu 18.04 (x86_64) server: Ascend-amct-{software version}-ubuntu18.04.x86_64.tar.gz
- EulerOS (AArch64) server: Ascend-amct-{software version}-euleros2.9.aarch64.tar.gz

{software version} indicates the version number.

#### 4.2.2 Pre-installation Actions

...
4.2.2.1 Ubuntu (x86)

Preparing the AMCT User

Any user (root or non-root) is allowed to install AMCT. This section uses a non-root user as an example.

- To install AMCT as the root user, skip this section.
- To install AMCT as an existing non-root user, ensure that the user has the read, write, and execute permissions on the $HOME directory.
- To install AMCT as a new non-root user, perform the following steps as the root user. The following uses this scenario as an example.
  a. Create an AMCT installation user and set the $HOME directory for the user:
     ```bash
     useradd -d /home/username -m username
     ```
  b. Set the user password:
     ```bash
     passwd username
     ```

  **NOTE**

  `username` indicates the name of the AMCT installation user. The `umask` value of the user is at least 0027.
  - You can view the `umask` value by running the `umask` command.
  - You can change the `umask` value by running the `umask NewValue` command.

(Optional) Setting the Permission of the AMCT Installation User

Skip this part if you install AMCT as the root user.

Before installing AMCT, you need to download the dependencies, which requires the `sudo apt-get` permission. Run the following commands as the root user:

1. Open the `/etc/sudoers` file:
   ```bash
   chmod u+w /etc/sudoers
   vi /etc/sudoers
   ```
2. Add the following content under `# User privilege specification` in the file:
   ```bash
   username ALL=(ALL:ALL) NOPASSWD:SETENV:/usr/bin/apt-get,/usr/bin/pip, /bin/tar, /bin/mkdir, /bin/sh, /bin/bash, /usr/bin/make, /usr/bin/pip3, /usr/bin/pip3.7, /usr/bin/pip3.7.5, /bin/ln
   ```
   Replace `username` with the name of the non-root user who executes the installation script.

   **NOTE**

   Check if the last line in the `/etc/sudoers` file is `#includedir /etc/sudoers.d`. If no, add it manually.

3. Run the `:wq!` command to save the file.
4. Remove the write permission on the `/etc/sudoers` file:
   ```bash
   chmod u-w /etc/sudoers
   ```
Setting Up Environment

**NOTE**

This tool supports Python 3.7.0 to Python 3.7.9. This document uses Python 3.7.5 as an example. The environment variables and installation commands vary according to the actual Python version.

AMCT runs on Ubuntu 18.04 (x86_64) and EulerOS (AArch64). The following table lists the architecture mapping of Ubuntu 18.04 (x86_64) servers.

**Table 4-4** Ubuntu (x86_64) architecture mapping

<table>
<thead>
<tr>
<th>Item</th>
<th>Version</th>
<th>How to Obtain</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Ubuntu 18.04</td>
<td>Click <a href="#">here</a> to download an Ubuntu release. The <code>ubuntu-18.04-server-amd64.iso</code> server install image is recommended.</td>
<td>-</td>
</tr>
<tr>
<td>Python</td>
<td>3.7.5</td>
<td>See <strong>4.9.4 Python 3.7.5 Installation on Ubuntu.</strong></td>
<td>Before running the <code>apt-get</code> command to install the dependencie s, ensure that the server can access the Internet.</td>
</tr>
<tr>
<td>PyTorch</td>
<td>1.4.0/1.5.0</td>
<td>Install a CPU or GPU version as needed. For details, see <a href="#">Installing Dependencies</a>.</td>
<td>-</td>
</tr>
<tr>
<td>CUDA Toolkit/CUDA Driver</td>
<td>10.1</td>
<td>Obtain required packages for installation. For example, you can obtain the Toolkit package from the following URL, which contains the Driver package. <a href="#">https://developer.nvidia.com/cuda-toolkit-archive</a></td>
<td>To perform GPU quantization, the CUDA software must be installed.</td>
</tr>
<tr>
<td>NumPy</td>
<td>1.16.0+</td>
<td>See <a href="#">Installing Dependencies</a>.</td>
<td>-</td>
</tr>
</tbody>
</table>
Checking Sources

During dependency installation, you need to make sure that the server of AMCT has Internet access. Run the following command as the root user to check source validity:

`apt-get update`

If an error is reported during the command execution, check whether the network connection is normal or replace the source in the `/etc/apt/sources.list` file with a valid one.

Installing Dependencies

Use the AMCT installation user to install software. If the installation user is a non-root user, run the `su - username` command to switch to the non-root user and run the following commands.

<table>
<thead>
<tr>
<th>Item</th>
<th>Version</th>
<th>How to Obtain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pillow</td>
<td>6.0.0+</td>
<td>-</td>
</tr>
<tr>
<td>Protobuf</td>
<td>3.11.0+</td>
<td>-</td>
</tr>
<tr>
<td>ONNX</td>
<td>1.6.0</td>
<td>-</td>
</tr>
<tr>
<td>ONNX Runtime</td>
<td>1.4.0</td>
<td>ONNX Runtime framework. For details, see Installing Dependencies.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Version</th>
<th>Installation Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>3.7.5</td>
<td>See 4.9.4 Python 3.7.5 Installation on Ubuntu.</td>
</tr>
</tbody>
</table>
## Install PyTorch

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Version</th>
<th>Installation Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU or GPU version of PyTorch</td>
<td>1.4.0/1.5.0</td>
<td>- CPU version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>python3.7.5 -m pip --trusted-host=download.pytorch.org install</td>
</tr>
<tr>
<td></td>
<td></td>
<td>torch==1.5.0+cpu torchvision==0.6.0+cpu -f <a href="https://download.pytorch.org/whl/torch_stable.html">https://download.pytorch.org/whl/torch_stable.html</a> --user</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- GPU version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>python3.7.5 -m pip --trusted-host=download.pytorch.org install</td>
</tr>
<tr>
<td></td>
<td></td>
<td>torch==1.5.0+cu101 torchvision==0.6.0+cu101 -f <a href="https://download.pytorch.org/whl/torch_stable.html">https://download.pytorch.org/whl/torch_stable.html</a> --user</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- CPU version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>python3.7.5 -m pip --trusted-host=download.pytorch.org install</td>
</tr>
<tr>
<td></td>
<td></td>
<td>torch==1.4.0+cpu torchvision==0.5.0+cpu -f <a href="https://download.pytorch.org/whl/torch_stable.html">https://download.pytorch.org/whl/torch_stable.html</a> --user</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- GPU version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>python3.7.5 -m pip --trusted-host=download.pytorch.org install</td>
</tr>
<tr>
<td></td>
<td></td>
<td>torch==1.4.0+cu100 torchvision==0.5.0+cu100 -f <a href="https://download.pytorch.org/whl/torch_stable.html">https://download.pytorch.org/whl/torch_stable.html</a> --user</td>
</tr>
</tbody>
</table>

### Python Environment

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumPy</td>
<td>1.16.0+</td>
<td>pip3.7.5 install numpy==1.16.0 --user</td>
</tr>
<tr>
<td>Pillow</td>
<td>6.0.0+</td>
<td>pip3.7.5 install pillow==6.0.0 --user</td>
</tr>
<tr>
<td>Protobuf</td>
<td>3.11.0+</td>
<td>pip3.7.5 install protobuf==3.11.0 --user</td>
</tr>
<tr>
<td>ONNNX</td>
<td>1.6.0</td>
<td>pip3.7.5 install onnx==1.6.0 --user</td>
</tr>
<tr>
<td>ONNX Runtime</td>
<td>1.4.0</td>
<td>pip3.7.5 install onnxruntime==1.4.0 --user</td>
</tr>
</tbody>
</table>

### Uploading the AMCT Package

Upload the `Ascend-cann-amct_{software version}_ubuntu18.04-x86_64.tar.gz` package to any directory (for example, `$HOME/amct/`) on the Linux server as the AMCT installation user.

Decompress the AMCT package:

```bash
tar -zxvf Ascend-cann-amct_{software version}_ubuntu18.04-x86_64.tar.gz
```

Find the following extracted packages.

**Table 4-6 Extracted parts of the AMCT package**

<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
<tbody>
<tr>
<td>amct caffe/</td>
<td>Caffe AMCT directory</td>
<td></td>
<td>OS support: Ubuntu 18.04 (x86_64)</td>
</tr>
<tr>
<td>Level-1 Directory</td>
<td>Level-2 Directory</td>
<td>Description</td>
<td>Use Case and Precaution</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Caffe AMCT</td>
<td>amct_caffe-{version}-py3-none-linux_{arch}.whl</td>
<td>Caffe AMCT package</td>
<td></td>
</tr>
</tbody>
</table>
  - For details, see AMCT Instructions (Caffe).
  - Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
|                   | amct_caffe_sample.tar.gz | Caffe quantization sample package | |
|                   | caffe_patch.tar.gz | Caffe patch package | |

| TensorFlow AMCT directory | amct_tensorflow-{version}-py3-none-linux_{arch}.whl | TensorFlow AMCT package |  
  - OS support: Ubuntu 18.04 (x86_64)
  - amct_tensorflow and amct_tensorflow_ascend cannot exist at the same time.
  - For details, see AMCT Instructions (TensorFlow).
  - Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
|                          | amct_tensorflow_sample.tar.gz | TensorFlow quantization sample package | |

| TF Adapter AMCT         | amct_tensorflow_ascend-{version}-py3-none-linux_{arch}.whl | TF Adapter AMCT package |  
  - OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment
  - amct_tensorflow and amct_tensorflow_ascend cannot exist at the same time.
  - For details, see AMCT Instructions (TensorFlow, Ascend).
  - Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
<p>|                         | amct_tensorflow_ascend_sample.tar.gz | Package of quantization samples using TF Adapter | |</p>
<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
</table>
| **amct_pytorch/** | PyTorch AMCT directory | | • OS support: Ubuntu 18.04 (x86_64)  
• For details, see AMCT Instructions (PyTorch).  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| amct_pytorch-{version}-py3-none-linux_{arch}.tar.gz | PyTorch AMCT source package | | |
| amct_pytorch_sample.tar.gz | PyTorch quantization sample package | | |
| **amct_onnx/** | ONNX AMCT directory | | • OS support: Ubuntu 18.04 (x86_64)  
• For details, see AMCT Instructions (ONNX).  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| amct_onnx-{version}-py3-none-linux_{arch}.whl | ONNX AMCT package | | |
| amct_onnx_op.tar.gz | ONNX Runtime AMCT custom OPP | | |
| amct_onnx_sample.tar.gz | ONNX quantization sample package | | |
| **amct_mindspore/** | MindSpore AMCT directory | | • OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
• For details, see AMCT Instructions (MindSpore).  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| amct_mindspore-{version}-py3-none-linux_{arch}.whl | MindSpore AMCT package | | |
| amct_mindspore_sample.tar.gz | MindSpore quantization sample package | | |
| **amct_acl/** | AscendCL API AMCT directory | | • OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
• For details, see AMCT Instructions (AscendCL).  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
<p>| Ascend-amct_acl-{software version}-{os}.{arch}.run | AscendCL API AMCT package | | |</p>
<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
<tbody>
<tr>
<td>amct_acl_sample.tar.gz</td>
<td>Package of quantization samples using AscendCL APIs</td>
<td>installed with the Ascend AI Processor.</td>
<td></td>
</tr>
</tbody>
</table>

{version} indicates the AMCT version number. {os}.{arch} indicates the OS and architecture.

### 4.2.3 Installation

**Step 1** In the directory where the AMCT package is located, run the following command to install the source code:

```bash
pip3.7.5 install amct_pytorch-{version}-py3-none-linux_{arch}.tar.gz --user
```

Replace `{version}` with the actual AMCT version number, and `{arch}` with the actual architecture of the installation server. If AMCT installation is performed by the root user and the --target option is included, ensure that the path specified by --target is the path of the current user.

**Step 2** Check the installation. If a message similar to the following is displayed, the installation is successful:

```
Successfully build amct-pytorch...
Successfully installed amct-pytorch-{version}
```

Find the installed AMCT in the python3.7.5 directory (for example, $HOME/.local/lib/python3.7.5/site-packages).

```
-rindir 5 amct amct 4096 Mar 17 11:50 amct_pytorch/
-rindir 2 amct amct 4096 Mar 17 11:50 amct_pytorch-{version}.dist-info/
```

amct_pytorch indicates the AMCT installation path.

----End

### 4.2.4 Post-installation Actions

You can set the AMCT quantization log level using environment variables. Logs include the logs printed to the screen and the logs saved in the amct_log/amct_pytorch.log file. The environment variables are optional. If they are not set, the default log level INFO is used.

- **Variables**
  
  The log level is set by the following variables:
  
  - **AMCT_LOG_FILE_LEVEL**: specifies the level of messages in the amct_pytorch.log file and the level of messages in the log file generated of the corresponding quantization layer when the model for accuracy simulation is generated.
  
  - **AMCT_LOG_LEVEL**: specifies the level of log messages printed to the screen.
Table 4-7 lists the valid values and their meanings.

<table>
<thead>
<tr>
<th>Logging Level</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG</td>
<td>Outputs DEBUG, INFO, WARNING, and ERROR logs.</td>
<td>Detailed process messages, including the quantization layer and corresponding processing phase (fusion, parameter quantization, or activation quantization)</td>
</tr>
<tr>
<td>INFO</td>
<td>Outputs INFO, WARNING, and ERROR logs. The default value is INFO.</td>
<td>Brief quantization processing messages, including the quantization phase.</td>
</tr>
<tr>
<td>WARNING</td>
<td>Outputs WARNING and ERROR logs.</td>
<td>Warning messages during quantization.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Outputs ERROR logs.</td>
<td>Error messages during quantization.</td>
</tr>
</tbody>
</table>

The logging level is case insensitive. That is, Info, info, and INFO are all valid values.

- **Examples**
  - The following commands are only examples. You can set the level as required.
  - Set the quantization log level of amct_pytorch.log to INFO.
    ```bash
    export AMCT_LOG_FILE_LEVEL=INFO
    ```
  - Set the level of the information displayed on the screen to INFO.
    ```bash
    export AMCT_LOG_LEVEL=INFO
    ```

4.3 Post-training Quantization

4.3.1 Quantization Preparations

Model

Upload the PyTorch model to be quantized to any directory on the Linux server as the AMCT installation user. The following uses the ResNet-101 model as an example.

If you choose to use your own model, you are advised to perform inference in the PyTorch environment in advance to test if it can run properly in the PyTorch environment with expected accuracy.

- **Download the model:**


a. Create the `resnet101_model` directory in any directory on the AMCT server, go to the directory, and run the following commands to obtain the model definition file and model checkpoint file:

```bash
mkdir resnet101_model
cd resnet101_model
# Obtain the model definition file.
wget https://raw.githubusercontent.com/pytorch/vision/v0.6.0/torchvision/models/resnet.py --no-check-certificate
# Obtain the model checkpoint file.
wget https://download.pytorch.org/models/resnet101-5d3b4d8f.pth --no-check-certificate
```

b. Create an empty `__init__.py` file in the `resnet101_model` directory for importing the Python module.

```bash
touch __init__.py
```

- **Modify the model definition file:**

  After the downloaded model definition file (`resnet.py`) is converted into an ONNX model, the flatten operator is introduced. However, this operator is not supported by the ATC tool. Therefore, you need to manually modify the model definition file to replace the flatten operator to the reshape operator as follows. Otherwise, model conversion using ATC fails.
  
a. Comment out the method of downloading the checkpoint file online. Instead, import the file from the local host. The following figure shows the model script before and after the tweaking.

**Figure 4-4 Model script before and after the tweaking**

The tweaked code lines are as follows.

```python
import os
CUR_DIR = os.path.split(os.path.realpath(__file__))[0]
```

b. Comment out the code for downloading checkpoint online in the `_resnet` function. Instead, import the file from the local host.

```python
model.load_state_dict(torch.load('{}/resnet101-5d3b4d8f.pth'.format(CUR_DIR)))
```

c. Comment out the invocation of the `torch.flatten` operator in the original model structure and use `torch.reshape()` instead. (Note: The shape of the reshape operator must be inferred online. Otherwise, the operator will be optimized to a flatten operator eventually according to the PyTorch architecture.)

The tweaked code line is as follows.

```python
x = torch.reshape(x, (x.shape[0], x.shape[1]))
```
Image Dataset

After the model is quantized using AMCT, perform inference with the model to test the model accuracy. Use the dataset that matches the model.

Upload the dataset matching the model to any directory on the Linux server as the AMCT installation user. The following uses the images dataset (which also serves as the calibration dataset) preset in the sample as an example.

Calibration Dataset

The calibration dataset is used to generate the quantization factors to guarantee the accuracy.

The process of calculating the quantization factors is referred to as calibration. Perform inference using the quantized network with one or more batches of a subset of images from the validation dataset to complete calibration.

Upload the calibration dataset file to any directory on the Linux server as the AMCT installation user.

4.3.2 Quantization Example

The following uses the image classification network's quantization script resnet101_sample.py, downloaded model, image dataset (images), and calibration dataset (images) to illustrate how to execute the quantization script.

1. Obtain the quantization script.

   In the directory of amct_pytorch_sample.tar.gz, extract the quantization script from the package:
   ```
   tar -zxvf amct_pytorch_sample.tar.gz
   cd sample
   ```

   Find the following extracted files and directories:
   - resnet101/resnet101_sample.py: post-training quantization sample script for the image classification network.
   - resnet101/images/: image dataset (also the calibration dataset) for the image classification network, containing 160 images.
   - resnet101/resnet101_retrain_sample.py: quantization aware training sample script for the image classification network. For details about the quantization workflow, see 4.4 Quantization Aware Training.
   - resnet101/retrain_conf: simplified quantization configuration file for quantization aware training.

2. Copy the model definition file prepared in Model to the sample/resnet101 directory (replace resnet101_model with the actual path).
   ```
   cp -r /home/hsisoc/PyTorch/model/resnet101_model ./resnet101
   ```

3. Run the quantization script.

   Run the following command in the sample/resnet101 directory to quantize the ResNet-101 network:
   ```
   python3.7.5 resnet101_sample.py
   ```
The preceding quantization command applies to single-GPU or non-GPU environments. If there are multiple GPUs, run the following command to specify the target GPU:
```
CUDA_VISIBLE_DEVICES=0 python3.7.5 resnet101_sample.py
```

If you want to use your own inference script to perform inference with the quantized model in the PyTorch environment, import the AMCT package by adding the following line to the beginning of the inference script:
```
import amct_pytorch
```

If messages similar to the following are displayed, the quantization is successful:
```
******final top1:0.8625
******final top5:0.9625  // Top 1 and top 5 inference accuracy results of the fake-quantized model in the ONNX Runtime environment.
[INFO] ResNet101 before quantize top1: 0.8875 top5: 0.9625  //Inference result of the original model. It is an example only.
[INFO] ResNet101 after quantize top1: 0.8625 top5: 0.9625  //Inference result of the fake-quantized model. It is an example only.
```

4. View the quantization result.

After the quantization is complete, find the following files generated in the directory of the quantization script:
- `results/calibration_results`: post-training quantization result directory.
  - `ResNet101_deploy_model.onnx`: quantized model file to be deployed on the Ascend AI Processor.
  - `ResNet101_fake_quant_model.onnx`: quantized model file that can be used for accuracy simulation in the ONNX Runtime environment.
- `tmp`: temporary directory, containing:
  - `config.json`: configuration file that describes how to quantize each layer in the model. If a quantization configuration file already exists in the directory of the quantization script, the existing quantization configuration file is overwritten by a new one with the same name in a call to `4.7.1.1 create_quant_config`. If not that case, a new quantization configuration file is created.

If the accuracy of model inference drops significantly after quantization, you can modify the `config.json` file by referring to `4.3.3 Quantization Configuration`.

- `modified_model.onnx`: ONNX model file exported after BN fusion is performed on the downloaded PyTorch model.
- `scale_offset_record.txt`: file that records quantization factors. For details about the prototype definition of the file, see `4.9.3 Quantization Factor Record File`.

5. (Optional) Convert the quantized model into an offline model adapted to the Ascend AI Processor by referring to `ATC Instructions`.
4.3.3 Quantization Configuration

This section describes the quantization configuration file of image classification networks.

4.3.3.1 Overview

If the inference accuracy of the config.json quantization configuration file generated by the 4.7.1.1 create_quant_config call does not meet the requirements, you need to tune the config.json file until the accuracy is as expected. The following is an example of the file content.

```json
{
  "version":1,
  "batch_num":2,
  "activation_offset":true,
  "do_fusion":true,
  "skip_fusion_layers":[],
  "conv":{
    "quant_enable":true,
    "activation_quant_params":{
      "max_percentile":0.999999,
      "min_percentile":0.999999,
      "search_range":[
        0.7,
        1.3
      ],
      "search_step":0.01
    },
    "weight_quant_params":{
      "channel_wise":true
    }
  },
  "layer1.0.conv1":{
    "quant_enable":true,
    "activation_quant_params":{
      "max_percentile":0.999999,
      "min_percentile":0.999999,
      "search_range":[
        0.7,
        1.3
      ],
      "search_step":0.01
    },
    "weight_quant_params":{
      "channel_wise":false
    }
  }
}
```

4.3.3.2 Configuration File Options

The following tables describe the parameters in the configuration file.

<table>
<thead>
<tr>
<th>Function</th>
<th>Version number of the quantization configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>1</td>
</tr>
</tbody>
</table>
Currently, only version 1 is available.

<table>
<thead>
<tr>
<th>Description</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recommended Value</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 4-9 batch_num**

<table>
<thead>
<tr>
<th>Function</th>
<th>Batch count for quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>int</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>Greater than 0</td>
</tr>
</tbody>
</table>
| **Description**  | If this option is not set, the default value 1 is used. It is recommended that the number of images in the calibration dataset be less than or equal to 50. The value of batch_num is calculated based on the value of batch_size.

batch_num \times batch_size equals the number of images in the calibration dataset used for quantization.

batch_size indicates the number of images per batch. |
| **Recommended Value** | 1          |
| **Required/Optional** | Optional        |

**Table 4-10 activation_offset**

<table>
<thead>
<tr>
<th>Function</th>
<th>Symmetric quantization or asymmetric quantization select for activation quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td><strong>true</strong> or <strong>false</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If it is set to <strong>true</strong>, asymmetric quantization is used. If it is set to <strong>false</strong>, symmetric quantization is used.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td><strong>true</strong></td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
<tr>
<td><strong>Table 4-11</strong> do_fusion</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td>Fusion switch</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>true or false</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If it is set to true, fusion is enabled. If it is set to false, fusion is disabled. Currently, only Conv+BN fusion is supported.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Table 4-12</strong> skip_fusion_layers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
</tr>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Table 4-13</strong> layer_config</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
</tr>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
</tr>
</tbody>
</table>
| **Description** | Includes the following parameters:  
  ● quant_enable  
  ● activation_quant_params  
  ● weight_quant_params |
<p>| <strong>Recommended Value</strong> | None |</p>
<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Optional</th>
</tr>
</thead>
</table>

**Table 4-14 quant_enable**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization enable by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td>If it is set to true, the layer is to be quantized. If it is set to false, otherwise.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>true</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 4-15 activation_quant_params**

<table>
<thead>
<tr>
<th>Function</th>
<th>Activation quantization parameters of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>Includes the following parameters:</td>
</tr>
<tr>
<td></td>
<td>● max_percentile</td>
</tr>
<tr>
<td></td>
<td>● min_percentile</td>
</tr>
<tr>
<td></td>
<td>● search_range</td>
</tr>
<tr>
<td></td>
<td>● search_step</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>None</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 4-16 weight_quant_params**

<table>
<thead>
<tr>
<th>Function</th>
<th>Weight quantization parameters of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
</tbody>
</table>
### channel_wise

<table>
<thead>
<tr>
<th>Description</th>
<th>Includes the following parameter: channel_wise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recommended Value</td>
<td>None</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

#### max_percentile

<table>
<thead>
<tr>
<th>Function</th>
<th>Upper search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0.5, 1]</td>
</tr>
<tr>
<td>Description</td>
<td>Indicates the maximum number to be considered as the search result among a group of numbers in descending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the maximum, that is, the largest number. A larger value indicates that the upper clip limit is closer to the maximum value of the data to be quantized.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>0.999999</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

#### min_percentile

<table>
<thead>
<tr>
<th>Function</th>
<th>Lower search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0.5, 1]</td>
</tr>
<tr>
<td>Description</td>
<td>Indicates the minimum number to be considered as the search result among a group of numbers in ascending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the minimum, that is, the smallest number. A larger value indicates that the lower clip limit is closer to the minimum value of the data to be quantized.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>0.999999</td>
</tr>
</tbody>
</table>

Copyright © Huawei Technologies Co., Ltd.
<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Optional</th>
</tr>
</thead>
</table>

**Table 4-19 search_range**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search range: [search_range_start, search_range_end]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>A list of two floats</td>
</tr>
<tr>
<td>Value Range</td>
<td>0 &lt; search_range_start &lt; search_range_end</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the quantization factor search range:</td>
</tr>
<tr>
<td></td>
<td>● search_range_start: search start.</td>
</tr>
<tr>
<td></td>
<td>● search_range_end: search end.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>[0.7, 1.3]</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 4-20 search_step**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0, (search_range_end – search_range_start)]</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the quantization factor search step. A smaller value indicates a smaller step.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>0.01</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 4-21 channel_wise**

<table>
<thead>
<tr>
<th>Function</th>
<th>Whether to use different quantization factors for each channel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
</tbody>
</table>

CANN
Development Auxiliary Tool Guide (Training)
4 AMCT Instructions (PyTorch)

Issue 01 (2021-07-01) Copyright © Huawei Technologies Co., Ltd. 140
4.3.3.3 Configuration Tuning

If the inference accuracy of the model quantized based on the default configuration in the config.json file drops significantly, perform the following steps to tune the quantization configuration file:

**Step 1** Execute the quantization script in the amct_pytorch_sample.tar.gz package to perform quantization based on the default configuration generated by the 4.7.1.1 create_quant_config API.

**Step 2** If the inference accuracy with the model quantized in **Step 1** is as expected, configuration tuning ends. Otherwise, go to **Step 3**.

**Step 3** Tune batch_num in the quantization configuration file.

`batch_num` controls the batch count for quantization. Tune it based on the batch size and the number of images required for quantization. Generally, a larger quantity of data samples used in a quantization process indicates a smaller accuracy loss after quantization. However, excessive data does not necessarily improve accuracy, but certainly consumes more memory and reduces the quantization speed, hence resulting in insufficient memory, video RAM, and thread resources. Therefore, it is recommended that the product of `batch_num` and `batch_size` be 16 or 32.

**Step 4** If the inference accuracy with the model quantized in **Step 3** is as expected, configuration tuning ends. Otherwise, go to **Step 5**.

**Step 5** Tune quant_enable in the quantization configuration file.

`quant_enable` specifies whether to quantize a layer. If set to `true`, the layer is to be quantized. If set to `false`, otherwise. If the configuration of a layer is not present, the quantization of the layer is skipped. Generally, specifying fewer layers to quantize improves quantization accuracy. When the network accuracy is not as expected, locate the quantization-sensitive layers (whose error increases significantly after quantization, such as the top layer, bottom layer, depthwise convolutional layer, and layers with few parameters) in the network, and disable quantization on these layers as needed.

**Step 6** If the inference accuracy with the model quantized in **Step 5** is as expected, configuration tuning ends. Otherwise, go to **Step 7**.

**Step 7** Tune the values of activation_quant_params and weight_quant_params in the quantization configuration file.

---

### Description

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>● If it is set to <strong>true</strong>, channels are separately quantized using different quantization factors.</td>
</tr>
<tr>
<td>● If it is set to <strong>false</strong>, all channels are quantized altogether using the same quantization factors.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommended Value</th>
<th>true</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Optional</th>
</tr>
</thead>
</table>
• Data is clipped to the range \([left, right]\) specified by the \texttt{activation\_quant\_params} parameters. Generally, values distributed near a boundary are sparse, and clip may be performed on all the values, to improve the accuracy. A larger value of \texttt{min\_percentile (max\_percentile)} indicates that \texttt{left (right)} is closer to the minimum value (maximum value) of the to-be-quantized data. \texttt{search\_range} and \texttt{search\_step} affect the range of \([left, right]\). Generally, a larger value of \texttt{search\_range} and a smaller value of \texttt{search\_step} may achieve higher quantization accuracy, but the quantization takes more time.

• \texttt{channel\_wise} in \texttt{weight\_quant\_params} determines whether to use a different quantization factor for each channel during weight quantization. If set to \texttt{true}, channels are separately quantized using different quantization factors. If set to \texttt{false}, all channels are quantized altogether using the same quantization factors. Generally, the inference accuracy is higher if the channels are separately quantized. However, the \texttt{torch.nn.Linear} layer is channel-irrelevant. Therefore, this parameter does not take effect on the layer.

\textbf{Step 8} If the inference accuracy with the model quantized in \textbf{Step 7} is as expected, configuration tuning ends. Otherwise, it indicates that quantization has severe adverse impact on the inference accuracy. In this case, remove the quantization configuration.

----End
4.4 Quantization Aware Training

Figure 4-5 Configuration tuning workflow

Start

Use the default quantization configuration

Is the accuracy acceptable?

Y

Tune batch count using batch_num.

N

Tune quantization layers using quant_enable.

Is the accuracy acceptable?

Y

Tune activation_quant_params and weight_quant_params.

N

Cancel quantization

End
4.4.1 Quantization Example

Prerequisites

- **Model**
  a. See Model.
  b. Copy the model files in Model to the sample/resnet101 directory (replace resnet101_model with the actual directory). If the files have been copied to the resnet101 directory, skip this step.

```
cp -r /home/hisisoc/PyTorch/model/resnet101_model ./resnet101
```

- **Image dataset**

Because quantization aware training needs huge data to further optimize the quantization parameters, keep the data consistent with that used to train the model. The ResNet-101 dataset is trained on the subset ILSVRC-2012-CLS of ImageNet. Therefore, you need to prepare a dataset in PyTorch format. Click here for more information. To use a different dataset, you need to preprocess the data by yourself.

Procedure

   Run the following command in the sample/resnet101 directory:
   - Single-device quantization aware training
     ```
     CUDA_VISIBLE_DEVICES=0 python3.7.5 resnet101_retrain_sample.py --train_set TRAIN_SET --eval_set EVAL_SET --config_defination CONFIG_DEFINATION --train_iter ITERATION
     ```
   - Multi-device quantization aware training
     ```
     CUDA_VISIBLE_DEVICES=0,1,2,3,4,5,6,7 python3.7.5 resnet101_retrain_sample.py --train_set TRAIN_SET --eval_set EVAL_SET --config_defination CONFIG_DEFINATION --train_iter ITERATION --distributed
     ```

   **NOTE**
   When multiple devices are used, only distributed training is supported. Data parallel training is not supported. In data parallel multi-device training, the following error is displayed:
   ```
   RuntimeError: Output 54 of BroadcastBackward is a view and its base or another view of its base has been modified inplace. This view is the output of a function that returns multiple views. Such functions do not allow the output views to be modified inplace. You should replace the inplace operation by an out-of-place one.
   ```

The following table describes the available command-line options.

<table>
<thead>
<tr>
<th>Table 4-22 Quantization aware training command-line options</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
</tbody>
</table>
| -h | • Optional.  
• Displays help information. |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--config_defination CONFIG_DEFINITION</code></td>
<td>• Optional.</td>
</tr>
<tr>
<td></td>
<td>• A string.</td>
</tr>
<tr>
<td></td>
<td>• Default: <strong>None</strong></td>
</tr>
<tr>
<td></td>
<td>• Sets the directory of the simplified</td>
</tr>
<tr>
<td></td>
<td>quantization configuration file.</td>
</tr>
<tr>
<td><code>--batch_num BATCH_NUM</code></td>
<td>• Optional.</td>
</tr>
<tr>
<td></td>
<td>• An int.</td>
</tr>
<tr>
<td></td>
<td>• Default: 2</td>
</tr>
<tr>
<td></td>
<td>• Sets the batch count in the</td>
</tr>
<tr>
<td></td>
<td>inference phase of quantization</td>
</tr>
<tr>
<td></td>
<td>aware training.</td>
</tr>
<tr>
<td><code>--train_set TRAIN_SET</code></td>
<td>• Required.</td>
</tr>
<tr>
<td></td>
<td>• A string.</td>
</tr>
<tr>
<td></td>
<td>• Default: <strong>None</strong></td>
</tr>
<tr>
<td></td>
<td>• Sets the directory of the image dataset.</td>
</tr>
<tr>
<td><code>--eval_set EVAL_SET</code></td>
<td>• Required.</td>
</tr>
<tr>
<td></td>
<td>• A string.</td>
</tr>
<tr>
<td></td>
<td>• Default: <strong>None</strong></td>
</tr>
<tr>
<td></td>
<td>• Sets the directory of the validation dataset.</td>
</tr>
<tr>
<td><code>--num_parallel_reads NUM_PARALLEL_READS</code></td>
<td>• Optional.</td>
</tr>
<tr>
<td></td>
<td>• An int.</td>
</tr>
<tr>
<td></td>
<td>• Default: 4</td>
</tr>
<tr>
<td></td>
<td>• Sets the number of threads for reading datasets.</td>
</tr>
<tr>
<td></td>
<td>Set the argument according to the hardware</td>
</tr>
<tr>
<td></td>
<td>compute capability.</td>
</tr>
<tr>
<td><code>--batch_size BATCH_SIZE</code></td>
<td>• Optional.</td>
</tr>
<tr>
<td></td>
<td>• An int.</td>
</tr>
<tr>
<td></td>
<td>• Default: <strong>25</strong></td>
</tr>
<tr>
<td></td>
<td>• Sets the batch size of PyTorch</td>
</tr>
<tr>
<td></td>
<td>execution. Set the argument according to the</td>
</tr>
<tr>
<td></td>
<td>memory or video RAM capacity.</td>
</tr>
<tr>
<td><code>--learning_rate LEARNING_RATE</code></td>
<td>• Optional.</td>
</tr>
<tr>
<td></td>
<td>• A float.</td>
</tr>
<tr>
<td></td>
<td>• Default: <strong>1e-5</strong></td>
</tr>
<tr>
<td></td>
<td>• Sets the learning rate.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| --train_iter TRAIN_ITER| • Optional.  
• An int.  
• Default: 2000  
• Sets the number of training iterations. |
| --print_freq PRINT_FREQ| • Optional.  
• An int.  
• Default: 10  
• Sets the frequency of printing training and test information. |
| --dist_url DIST_URL    | • Optional.  
• A string.  
• Default: tcp://127.0.0.1:50011  
• Sets the method of initializing the multi-device training communication process. |
| --distributed         | • Optional.  
• Data type: none  
• Specifies whether to perform multi-device training. |

An example is as follows.

CUDA_VISIBLE_DEVICES=0 python3.7.5 resnet101_retrain_sample.py --train_set /data/Datasets/imagenetPytorch/train --eval_set /data/Datasets/imagenetPytorch/eval --config_defination ./retrain_conf/retrain.cfg

If messages similar to the following are displayed, the quantization aware training is successful:

[INFO] ResNet101 before retrain top1:77.37% top5:93.55%  
[INFO] ResNet101 after retrain top1:77.42% top5:93.56%

2. View the retrain result.

After the quantization aware training is complete, find the following folders and files generated to the sample/resnet101 directory:

- **amct_log/amct_pytorch.log**: AMCT log file.
- **results/retrain_results**: result directory of the quantization aware training.
  - **ResNet101_fake_quant_model.onnx**: quantized model file that can be used for accuracy simulation in the ONNX Runtime environment.
  - **ResNet101_deploy_model.onnx**: quantized model file to be deployed on the Ascend AI Processor.
- **tmp**: quantization aware training temporary directory, containing:
- **config.json**: quantization aware training configuration file that describes how to train each layer in the model. If a quantization aware configuration file already exists in the directory of the quantization aware script, the existing configuration file is overwritten by a new one with the same name in a call to 4.7.2.1 `create_quant_retrain_config`. If not that case, a new configuration file is created.

- **model_best.pth.tar**: checkpoint files generated during PyTorch quantization aware training.

- **record.txt**: file that records quantization factors. For details about the prototype definition of the file, see 4.9.3 Quantization Factor Record File.

When quantization aware training is run again on the model, the existing result files will be overwritten.

### 4.4.2 Quantization Configuration

#### 4.4.2.1 Overview

If inference based on the **config.json** quantization aware training configuration file generated by the 4.7.2.1 `create_quant_retrain_config` call has significant accuracy drop, tune the **config.json** file until the accuracy is as expected. The following is an example of the file content. Keep the layer names unique in the JSON file.

```json
{
   "version":1,
   "batch_num":1,
   "conv":{
      "retrain_enable":true,
      "retrain_data_config":{
         "algo":"ulq_quantize"
      },
      "retrain_weight_config":{
         "algo":"arq_retrain",
         "channel_wise":true
      }
   },
   "layer1.0.conv1":{
      "retrain_enable":true,
      "retrain_data_config":{
         "algo":"ulq_quantize"
      },
      "retrain_weight_config":{
         "algo":"arq_retrain",
         "channel_wise":true
      }
   },
   "fc":{
      "retrain_enable":true,
      "retrain_data_config":{
         "algo":"ulq_quantize"
      },
      "retrain_weight_config":{
         "algo":"arq_retrain",
         "channel_wise":false
      }
   }
}```
4.4.2.2 Configuration File Options

The following describes the configuration options available in the configuration file. Note that Table 4-30 to Table 4-32 are available only when you manually tune the quantization configuration file.

**Table 4-23 version**

<table>
<thead>
<tr>
<th>Function</th>
<th>Version number of the quantization configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>Currently, only version 1 is available.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>1</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 4-24 batch_num**

<table>
<thead>
<tr>
<th>Function</th>
<th>Batch count in the inference phase of quantization aware training.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>Greater than 0</td>
</tr>
<tr>
<td>Description</td>
<td>If this option is not set, the default value 1 is used. It is recommended that the number of images in the calibration dataset be less than or equal to 50. The value of batch_num is calculated based on the value of batch_size. batch_num x batch_size equals the number of images in the calibration dataset used for quantization. batch_size indicates the number of images per batch.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>1</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 4-25 retrain_enable**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization aware training enable by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
</tbody>
</table>
If set to **true**, quantization aware training is performed at this layer. If set to **false**, otherwise.

<table>
<thead>
<tr>
<th>Description</th>
<th>If set to true, quantization aware training is performed at this layer. If set to false, otherwise.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recommended Value</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 4-26 retrain_data_config**

<table>
<thead>
<tr>
<th>Function</th>
<th>Activation quantization configuration by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>object</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
| **Description** | Includes the following parameters:  
- algo: selects the quantization algorithm, default to ulq_quantize.  
- clip_max: sets the upper limit of the clip quantization algorithm, default to be empty.  
- clip_min: sets the lower limit of the clip quantization algorithm, default to be empty.  
- fixed_min: fixes the minimum value of the clip quantization algorithm at 0, default to be empty. |
| **Recommended Value** | None |
| **Required/Optional** | Optional |

**Table 4-27 retrain_weight_config**

<table>
<thead>
<tr>
<th>Function</th>
<th>Weight quantization configuration by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>object</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
| **Description** | Includes the following parameters:  
- algo: quantization algorithm select, default to arq_retrain.  
- channel_wise |
| **Recommended Value** | None |
| **Required/Optional** | Optional |
Table 4-28 algo

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization algorithm by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
</tbody>
</table>
| Description               | ● ulq_quantize: ULQ clip quantization algorithm.  
|                           | ● arq_retrain: ARQ quantization algorithm.        |
| Recommended Value         | Set to ulq_quantize for activation quantization or arq_retrain for weight quantization. |
| Required/Optional         | Optional                         |

Table 4-29 channel_wise

<table>
<thead>
<tr>
<th>Function</th>
<th>Whether to use different quantization factors for each channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
</tbody>
</table>
| Description               | ● If set to true, channels are separately quantized using different quantization factors.  
|                           | ● If set to false, all channels are quantized altogether using the same quantization factors. |
| Recommended Value         | true                                                          |
| Required/Optional         | Optional                                                       |

Table 4-30 fixed_min

<table>
<thead>
<tr>
<th>Function</th>
<th>Lower limit enable of the activation quantization algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>● If set to <strong>true</strong>, the lower limit of the activation quantization algorithm is fixed at 0.</td>
<td></td>
</tr>
<tr>
<td>● If set to <strong>false</strong>, the lower limit of the activation quantization algorithm is not fixed.</td>
<td></td>
</tr>
<tr>
<td>If this option is not included, AMCT automatically sets the lower limit of the activation quantization algorithm according to the graph structure.</td>
<td></td>
</tr>
<tr>
<td>If this option is included: when the upstream layer of the quantization layer is ReLU, you need to manually set this option to <strong>true</strong>; when the upstream layer of the quantization layer is not ReLU, you need to manually set this option to <strong>false</strong>.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommended Value</th>
<th>Do not include this option.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Optional</th>
</tr>
</thead>
</table>

**Table 4-31 clip_max**

<table>
<thead>
<tr>
<th>Function</th>
<th>Upper limit of the activation quantization algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>clip_max&gt;0</td>
</tr>
</tbody>
</table>

Controls the upper limit **max** based on the data distribution of the activations at different layers. The recommended value range is as follows: 

\[0.3 \times \text{max}, 1.7 \times \text{max}\]

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>If this option is included, the clip upper limit of the activation quantization algorithm is fixed. If this option is not included, the clip upper limit is learned using the IFMR algorithm.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommended Value</th>
<th>Do not include this option.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Optional</th>
</tr>
</thead>
</table>

**Table 4-32 clip_min**

<table>
<thead>
<tr>
<th>Function</th>
<th>Lower limit of the activation quantization algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>clip_min&lt;0</td>
</tr>
</tbody>
</table>

Controls the lower limit **min** based on the data distribution of the activations at different layers. The recommended value range is as follows: 

\[0.3 \times \text{min}, 1.7 \times \text{min}\]
<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>If this option is included, the clip lower limit of the activation quantization algorithm is fixed. If this option is not included, the clip lower limit is learned using the IFMR algorithm.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recommended Value</strong></td>
<td>Do not include this option.</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

### 4.4.2.3 Configuration Tuning

If the inference accuracy of the model quantized based on the default `config.json` configuration drops significantly, perform the following steps to tune the quantization configuration file.

1. Execute the quantization script in the `amct_pytorch_sample.tar.gz` package to perform quantization based on the default configuration generated by the `4.7.2.1 create_quant_retrain_config` call. If the quantization accuracy is as expected, the configuration tuning ends. Otherwise, go to 2.

2. Cancel the quantization of certain layers by changing the value of `retrain_enable` to `false`. Generally, the input and output layers of a model have a greater impact on the inference result. Therefore, you can try to cancel the quantization of the input and output layers first.

   If you have desirable settings for `clip_max` and `clip_min`, modify the quantization configuration file as follows.

   ```json
   {
   "version":1,
   "batch_num":1,
   "layername1":{
   "retrain_enable":true,
   "retrain_data_config":{
   "algo":"ulq_quantize",
   "clip_max":3.0,
   "clip_min":-3.0
   },
   "retrain_weight_config":{
   "algo":"arq_retrain",
   "channel_wise":true
   }
   },
   "layername2":{
   "retrain_enable":true,
   "retrain_data_config":{
   "algo":"ulq_quantize",
   "clip_max":3.0,
   "clip_min":-3.0
   },
   "retrain_weight_config":{
   "algo":"arq_retrain",
   "channel_wise":true
   }
   }
   }
   ```

3. Configuration tuning ends if the inference accuracy meets the requirement. Otherwise, it indicates that quantization aware training has severe adverse impact on the inference accuracy. In this case, remove the quantization aware training configuration.
4.5 AMCT Update

The latest AMCT release allows you to access to the latest features. Before updating AMCT, uninstall the existing installation by referring to 4.6 AMCT Uninstallation, and then install the latest version by referring to 4.2 AMCT Installation.

4.6 AMCT Uninstallation

You can uninstall AMCT as follows:

1. Run the following command in any directory on the Linux server as the AMCT installation user:
   ```
   pip3.7.5 uninstall amct_pytorch
   ```
2. When the following information is displayed, enter y:
   ```
   Uninstalling amct-pytorch-{version}:
   Would remove:
   ...
   Proceed (y/n)? y
   ```
   If a message similar to the following is displayed, the uninstallation is successful:
   ```
   Successfully uninstalled amct-pytorch-{version}
   ```

4.7 APIs

4.7.1 Post-training Quantization

4.7.1.1 create_quant_config

Description

Applies to post-training quantization. Finds all quantization-capable layers in a graph, creates a quantization configuration file, and writes the quantization configuration information of the quantization-capable layers to the file.

Prototype

```
create_quant_config(config_file, model, input_data, skip_layers=None, batch_num=1, activation_offset=True, config_defination=None)
```
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the quantization configuration file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The existing file (if available) in the directory will be overwritten upon this API call.</td>
<td></td>
</tr>
<tr>
<td>model</td>
<td>Input</td>
<td>Model to be quantified, with weights loaded.</td>
<td>A torch.nn.module.</td>
</tr>
<tr>
<td>input_data</td>
<td>Input</td>
<td>Input data of a model. A torch.tensor, equivalent to a tuple(torch.tensor).</td>
<td>A tuple.</td>
</tr>
<tr>
<td>skip_layers</td>
<td>Input</td>
<td>Quantization-capable layers to skip.</td>
<td>Default: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Restriction: If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect.</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Input / Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| batch_num    | Input          | Number of batches taken to generate the quantization factors.               | An int.                                                                                                                                                    Value range: any integer larger than 0.                                                                 Default: 1                                                   Restrictions:  
  - batch_num cannot be too large. The product of batch_num and batch_size equals to the number of images used during quantization. Too many images consume too much memory.  
  - If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect. |
| activation_offset | Input          | Whether to quantize activations with offset.                               | Default: true                                                                                                                                             A bool.                                                                 Restriction: If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect. |
| Param
eter | Input
/ Return | Description | Restriction |
|---------|-----------|-------------|-------------|
| config
definition | Input | Whether to create a simplified quantization configuration file `quant.cfg` from the `calibration_config_pytorch.proto` file in `/amct_pytorch/proto/calibration_config_pytorch.proto` under the AMCT installation path. For details about the parameters in the `calibration_config_pytorch.proto` template and the generated simplified quantization configuration file `quant.cfg`, see 4.9.1 Simplified Post-training Quantization Configuration File. | Default: None
A string.
Restriction: If it is set to None, a configuration file is generated based on the residual arguments (skip_layers, batch_num, and activation_offset). Otherwise, a configuration file in JSON format is generated based on this argument. | |

**Returns**

None

**Outputs**

Outputs a quantization configuration file in JSON format. (When quantization is performed again, the existing one output by this API call will be overwritten.)

```
{
  "version":1,
  "batch_num":2,
  "activation_offset":true,
  "do_fusion":true,
  "skip_fusion_layers":[],
  "conv":{
    "quant_enable":true,
    "activation_quant_params":{
      "max_percentile":0.999999,
      "min_percentile":0.999999,
      "search_range":[
        0.7,
        1.3
      ],
      "search_step":0.01
    },
    "weight_quant_params":{
      "channel_wise":true
    }
  },
  "layer1.0.conv1":{
    "quant_enable":true,
```
CANN
Development Auxiliary Tool Guide (Training)

}

}

4 AMCT Instructions (PyTorch)

"activation_quant_params":{
"max_percentile":0.999999,
"min_percentile":0.999999,
"search_range":[
0.7,
1.3
],
"search_step":0.01
},
"weight_quant_params":{
"channel_wise":false
}

Example
import amct_pytorch as amct
# Create a graph of the network to be quantized.
model = build_model()
model.load_state_dict(torch.load(state_dict_path))
input_data = tuple([torch.randn(input_shape)])
model.eval()
# Create a quantization configuration file.
amct.create_quant_config(config_file="./configs/config.json",
model=model,
input_data=input_data,
skip_layers=None,
batch_num=1,
activation_offset=True)

4.7.1.2 quantize_model
Description
Applies to post-training quantization. Quantizes a graph based on the
quantization configuration file, inserts the quantization operators, generates a
quantization factor record file record_file, and returns the calibrated model of the
torch.nn.module type.

Prototype
calibration_model = quantize_model(config_file, modfied_onnx_file,
record_file, model, input_data, input_names=None, output_names=None,
dynamic_axes=None)

Parameters
Parameter

Input/
Retur
n

Description

Restriction

config_file

Input

Quantization configuration file
generated by the user, which is used
to specify the configuration of the
quantization layer in the network.

A string.

Issue 01 (2021-07-01)

Copyright © Huawei Technologies Co., Ltd.

157


<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>modified_onnx_file</td>
<td>Input</td>
<td>File name of the result ONNX model.</td>
<td>A string.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td>model</td>
<td>Input</td>
<td>Model to be quantified, with weights loaded.</td>
<td>A torch.nn.module.</td>
</tr>
<tr>
<td>input_data</td>
<td>Input</td>
<td>Input data of a model. A torch.tensor, equivalent to a tuple(torch.tensor).</td>
<td>A tuple.</td>
</tr>
<tr>
<td>input_names</td>
<td>Input</td>
<td>Input names of the model, which are used in modified_onnx_file.</td>
<td>Default: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A list of strings.</td>
<td>A list of strings.</td>
</tr>
<tr>
<td>output_names</td>
<td>Input</td>
<td>Output names of the model, which are used in modified_onnx_file.</td>
<td>Default: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A list of strings.</td>
<td></td>
</tr>
<tr>
<td>dynamic_axes</td>
<td>Input</td>
<td>Dynamic axes of the model inputs and outputs. For example, if the inputs have format NCHW, where N, H and W are uncertain, and the outputs have format NL, where N is uncertain, then: {&quot;inputs&quot;: [0,2,3], &quot;outputs&quot;: [0]}.</td>
<td>Default: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A dict&lt;string, dict&lt;python:int, string&gt;&gt;, or dict&lt;string, list&lt;int&gt;&gt;</td>
<td></td>
</tr>
<tr>
<td>calibration_model</td>
<td>Return</td>
<td>Result calibrated model of the torch.nn.module type</td>
<td>Default: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A torch.nn.module.</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

Result calibrated model of the torch.nn.module type.

**Example**

```python
calibration_model = amct.quantize_model(config_json_file, modified_model,
    scale_offset_record_file, model,
    input_data,
    input_names=['input'],
    output_names=['output'],
```

```python
import amct_pytorch as amct
# Create a graph of the network to be quantized.
model = build_model()
model.load_state_dict(torch.load(state_dict_path))
input_data = tuple([torch.randn(input_shape)])

scale_offset_record_file = os.path.join(TMP, 'scale_offset_record.txt')
modified_model = os.path.join(TMP, 'modified_model.onnx')
# Quantize the model.
calibration_model = amct.quantize_model(config_json_file, modified_model,
    scale_offset_record_file, model,
    input_data,
    input_names=['input'],
    output_names=['output'],
```
4.7.1.3 save_model

Description

Applies to post-training quantization. Saves a model file that can be used for both accuracy simulation in the ONNX Runtime environment and inference on the Ascend AI Processor based on the record_file quantization factor record file.

Restrictions

- This API is called after batch_num is reached. Otherwise, the quantization factors are incorrect and the quantization result is compromised.
- This API receives only the ONNX model file returned by 4.7.1.2 quantize_model.
- The quantization factor record file passed to the API call is generated in the 4.7.1.2 quantize_model phase. The factor values will be filled in the model inference phase.

Prototype

save_model(modified_onnx_file, record_file, save_path)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>modified_onnx_file</td>
<td>Input</td>
<td>File name of the result ONNX model.</td>
<td>A string.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, ./quantized_model/*model.</td>
<td>A string.</td>
</tr>
</tbody>
</table>

Returns

None

Outputs

- Outputs an ONNX model for accuracy simulation in the ONNX Runtime environment with the name containing the fake_quant keyword.
CANN
Development Auxiliary Tool Guide (Training)

●

4 AMCT Instructions (PyTorch)

Outputs a deployable ONNX model file with the name containing the deploy
keyword. The model can be deployed on the Ascend AI Processor after being
converted by the ATC tool.

When quantization is performed again, the existing files in the output directory
will be overwritten upon this API call.

Example
import amct_pytorch as amct
# Perform network inference and complete quantization during the inference.
for i in batch_num:
output = calibration_model(input_batch)
# Insert the API call and save the quantized model as an ONNX file.
amct.save_model(modfied_onnx_file="./tmp/modified_model.onnx",
record_file="./tmp/scale_offset_record.txt",
save_path="./results/model")

4.7.2 Quantization Aware Training
4.7.2.1 create_quant_retrain_config
Description
Applies to quantization aware training. Finds all quantization-capable layers in a
graph, creates a quantization configuration file, and writes the quantization
configuration information of the quantization-capable layers to the configuration
file.

Prototype
create_quant_retrain_config(config_file, model, input_data,
config_defination=None)

Parameters
Paramet
er

Input/
Retur
n

Description

Restriction

config_fil
e

Input

Directory of the configuration
file, including the file name.

A string.

The existing file (if available)
in the directory will be
overwritten upon this API call.
model

Input

Original model, with weights
loaded.

A torch.nn.module.

input_dat
a

Input

Input data of a model. A
torch.tensor, equivalent to a
tuple(torch.tensor).

A tuple.

Issue 01 (2021-07-01)

Copyright © Huawei Technologies Co., Ltd.

160


<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_defination</td>
<td>Input</td>
<td>Simplified quantization configuration file.</td>
<td>Default: None&lt;br&gt;A string. Restriction: If it is set to <strong>None</strong>, a configuration file is generated based on the residual arguments. Otherwise, a configuration file in JSON format is generated based on this argument.</td>
</tr>
</tbody>
</table>

**Returns**

None

**Outputs**

Outputs a quantization aware training configuration file in JSON format. The existing configuration file (if available) in the directory will be overwritten upon this API call. An example is as follows.

```json
{
    "version":1,
    "batch_num":1,
    "conv1":{
        "retrain_enable":true,
        "retrain_data_config":{
            "algo":"ulq_quantize"
        },
        "retrain_weight_config":{
            "algo":"arq_retrain",
            "channel_wise":true
        }
    },
    "layer1.0.conv1":{
        "retrain_enable":true,
        "retrain_data_config":{
            "algo":"ulq_quantize"
        }
    }
}
```
"retrain_weight_config":{
    "algo":"arq_retrain",
    "channel_wise":true
  },
  "fc":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo":"ulq_quantize"
    },
    "retrain_weight_config":{
      "algo":"arq_retrain",
      "channel_wise":false
    }
  }
},

Example

import amct_pytorch as amct
# Create a graph of the network to be quantized.
model = build_model()
model.load_state_dict(torch.load(state_dict_path))
input_data = tuple([torch.randn(input_shape)])

# Create a quantization configuration file.
amct.create_quant_retrain_config(config_file="./configs/config.json",
                                  model=model,
                                  input_data=input_data)

4.7.2.2 create_quant_retrain_model

Description

Applies to quantization aware training. Quantizes a graph based on the given configuration file, inserts quantization-related layers (quantization-aware layers of data and weights and layers for searching for N), generates a quantization factor record file (record_file), and saves the new model of the torch.nn.module type to a file.

Prototype

quant_retrain_model = create_quant_retrain_model (config_file, model, record_file, input_data)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input / Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization aware training configuration file generated by the user, which is used to specify the configuration of the quantization layer in the network.</td>
<td>A string.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input / Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>model</td>
<td>Input</td>
<td>Original model, with weights loaded.</td>
<td>A torch.nn.module.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td>input_data</td>
<td>Input</td>
<td>Input data of a model. A torch.tensor, equivalent to a tuple(torch.tensor).</td>
<td>A tuple.</td>
</tr>
<tr>
<td>quant_retrain_model</td>
<td>Return</td>
<td>Result model of the torch.nn.module type for quantization aware training.</td>
<td>Default: None</td>
</tr>
</tbody>
</table>

## Returns

Result model of the torch.nn.module type for quantization aware training.

## Outputs

None

## Example

```python
import amct_pytorch as amct
# Build a graph of the model for quantization aware training.
model = build_model()
model.load_state_dict(torch.load(state_dict_path))
input_data = tuple([torch.randn(input_shape)])

scale_offset_record_file = os.path.join(TMP, 'scale_offset_record.txt')
# Quantize the model.
quant_retrain_model = amct.create_quant_retrain_model(
    config_json_file,
    model, scale_offset_record_file, input_data)
```

### 4.7.2.3 restore_quant_retrain_model

## Description

Applies to quantization aware training. Quantizes a graph based on the given configuration file, inserts quantization-related layers (quantization-aware layers of data and weights and layers for searching for N), generates a quantization factor record file (*record_file*), loads the checkpoint weight parameters saved during training, and returns the result model of the torch.nn.module type.
Prototype

```python
quant_retrain_model = restore_quant_retrain_model (config_file, model, record_file, input_data, pth_file, state_dict_name=None)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization aware training configuration file generated by the user, which is used to specify the configuration of the quantization layer in the network. The <code>config.json</code> file passed to this call must be the same as that pass to the <code>create_quant_retrain_model</code> call.</td>
<td>A string.</td>
</tr>
<tr>
<td>model</td>
<td>Input</td>
<td>Original model, with weights unloaded.</td>
<td>A torch.nn.module.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td>input_data</td>
<td>Input</td>
<td>Input data of a model. A torch.tensor, equivalent to a tuple(torch.tensor).</td>
<td>A tuple.</td>
</tr>
<tr>
<td>pth_file</td>
<td>Input</td>
<td>Weight file saved during training.</td>
<td>A string.</td>
</tr>
<tr>
<td>state_dict_name</td>
<td>Input</td>
<td>Key value corresponding to the weight in the weight file.</td>
<td>Default: None</td>
</tr>
<tr>
<td>quant_retrain_model</td>
<td>Return</td>
<td>Result model of the torch.nn.module type for quantization aware training.</td>
<td>Default: None</td>
</tr>
</tbody>
</table>

Returns

Result model of the torch.nn.module type for quantization aware training.

Outputs

None

Example

```python
import amct_pytorch as amct
# Create a graph of the network to be quantized.
model = build_model()
```
import_data = tuple([torch.randn(input_shape)])

scale_offset_record_file = os.path.join(TMP, 'scale_offset_record.txt')

# Quantize the model.
quant_retrain_model = amct.restore_quant_retrain_model(
    config_json_file,
    model,
    scale_offset_record_file,
    input_data,
    pth_file)

4.7.2.4 save_quant_retrain_model

Description

Applies to quantization aware training. Saves a retrained model to two model files, one for accuracy simulation and the other for inference on the Ascend AI Processor.

Prototype

save_quant_retrain_model (config_file, model, record_file, save_path, input_data, input_names=None, output_names=None, dynamic_axes=None)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization aware training configuration file generated by the user, which is used to specify the configuration of the quantization layer in the network.</td>
<td>A string.</td>
</tr>
<tr>
<td>model</td>
<td>Input</td>
<td>Retrained model.</td>
<td>A torch.nn.module.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, ./quantized_model/*model.</td>
<td>A string.</td>
</tr>
<tr>
<td>input_data</td>
<td>Input</td>
<td>Input data of a model. A torch.tensor, equivalent to a tuple (torch.tensor).</td>
<td>A tuple.</td>
</tr>
<tr>
<td>input_names</td>
<td>Input</td>
<td>Input names of the result ONNX model. Default: None A list of strings.</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restrictions</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>output_names</td>
<td>Input</td>
<td>Output names of the result ONNX model.</td>
<td>Default: <strong>None</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A list of strings.</td>
</tr>
<tr>
<td>dynamic_axes</td>
<td>Input</td>
<td>Dynamic axes of the model inputs and outputs. For example, if the inputs have format NCHW, where N, H and W are uncertain, and the outputs have format NL, where N is uncertain, then: { &quot;inputs&quot;: [0,2,3], &quot;outputs&quot;: [0]}</td>
<td>Default: <strong>None</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A dict&lt;string, dict&lt;python:int, string&gt;&gt;, or dict&lt;string, list(int)&gt;</td>
</tr>
</tbody>
</table>

**Returns**

None

**Outputs**

- Outputs a model for accuracy simulation in the ONNX Runtime environment, with the name containing the **fake_quant** keyword.
- Outputs a deployable model file with the name containing the **deploy** keyword. The model can be deployed on the Ascend AI Processor after being converted by the ATC tool.

When quantization aware training is performed again, the existing files in the output directory will be overwritten upon this API call.

**Example**

```python
import amct_pytorch as amct
# Create a graph of the network to be quantized.
model = build_model()
model.load_state_dict(torch.load(state_dict_path))
input_data = tuple((torch.randn(input_shape)))
# Trained the retrained model to train quantization factors.
train_model(quant_retrain_model, input_batch)
# Infer with the retrained model to export the quantization factors.
infer_model(quant_retrain_model, input_batch)

# Insert the quantization API call and save the retrained model as an ONNX file.
amct.save_quant_retrain_model(
    config_json_file,
    model,
    scale_offset_record_file,
    input_data,
    input_names=['input'],
    output_names=['output'],
    dynamic_axes={'input':{0: 'batch_size'},
                  'output':{0: 'batch_size'}}
)
```
4.8 FAQ

4.8.1 An Error Message Is Displayed During python3-tk Installation

Symptom

When the python3-tk dependency is installed, the following error message is displayed.

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
Python3-tk is already the newest version (3.6.9-18.04).
The following packages were automatically installed and are no longer required:
  libboost-iostreams1.72.0 libboost-test1 1:7.1.0-7  libboost-thread1 1:1.72.0-7
We recommend you remove them.
```

Solution

Copy the missing file `py_compile.py` to the `/usr/lib/python3.7` directory and reinstall the Python.

```
cp /usr/local/python3.7.5/lib/python3.7/py_compile.py /usr/lib/python3.7
```

Replace `/usr/local/python3.7.5/lib/python3.7/py_compile.py` with the actual path of the file.

4.9 Appendixes

4.9.1 Simplified Post-training Quantization Configuration File

Table 4-33 describes the parameters in the `calibration_config_pytorch.proto` template.
<table>
<thead>
<tr>
<th>Message</th>
<th>Required / Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified post-training quantization configuration file of AMCT.</td>
</tr>
<tr>
<td>Optional</td>
<td>uint32</td>
<td>batch_num</td>
<td></td>
<td>Batch count used for quantization.</td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>activatior_offset</td>
<td></td>
<td>Whether to quantize activations with offset.</td>
</tr>
<tr>
<td>Repeated</td>
<td>string</td>
<td>skip_layers</td>
<td></td>
<td>Layers to skip during quantization.</td>
</tr>
<tr>
<td>Repeated</td>
<td>string</td>
<td>skip_layer_types</td>
<td></td>
<td>Types of layers to skip during quantization.</td>
</tr>
<tr>
<td>Optional</td>
<td>CalibrationConfig</td>
<td>common_config</td>
<td></td>
<td>Common quantization configuration. If a layer is not overridden by <code>override_layer_types</code> or <code>override_layer_configs</code>, this configuration is used.</td>
</tr>
<tr>
<td>Repeated</td>
<td>OverridingLayerType</td>
<td>override_layer_types</td>
<td></td>
<td>Type of layers to override.</td>
</tr>
<tr>
<td>Repeated</td>
<td>OverridingLayer</td>
<td>override_layer_configs</td>
<td></td>
<td>Layer to override.</td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>do_fusion</td>
<td></td>
<td>BN fusion switch. Defaults to <code>true</code>.</td>
</tr>
<tr>
<td>Repeated</td>
<td>string</td>
<td>skip_fusion_layer</td>
<td></td>
<td>Layers to skip in BN fusion.</td>
</tr>
<tr>
<td>OverrideLayerType</td>
<td>Required</td>
<td>string</td>
<td>layer_type</td>
<td>Type of layers to quantize.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Req</td>
<td>Calib</td>
<td>calibrati</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td></td>
<td>uired</td>
<td>tionConfig</td>
<td>on_config</td>
<td></td>
</tr>
<tr>
<td>OverrideLayer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization configuration overriding by layer.</td>
</tr>
<tr>
<td></td>
<td>Req</td>
<td>string</td>
<td>layer_name</td>
<td>Layer to override.</td>
</tr>
<tr>
<td></td>
<td>uired</td>
<td>Calib</td>
<td>calibrati</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td></td>
<td>uired</td>
<td>tionConfig</td>
<td>on_config</td>
<td></td>
</tr>
<tr>
<td>Calibration</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Calibration-based quantization configuration.</td>
</tr>
<tr>
<td>Config</td>
<td>-</td>
<td>ARQuantize</td>
<td>arq_quantize</td>
<td>Weight quantization algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>arq_quantize</strong>: ARQ algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>FMRQuantize</td>
<td>ifmr_quantize</td>
<td>Activation quantization algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>ifmr_quantize</strong>: IFMR algorithm configuration.</td>
</tr>
<tr>
<td>ARQuantize</td>
<td>-</td>
<td>bool</td>
<td>channel_wise</td>
<td>Whether to use different quantization factors for each channel.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FMRQuantize</td>
<td>-</td>
<td>float</td>
<td>search_range_start</td>
<td>FMR quantization algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>search_range_end</td>
<td>Quantization factor search start.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>search_step</td>
<td>Quantization factor search end.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The following is an example simplified quantization configuration file (*quant.cfg*).

```plaintext
# Global quantization parameters
batch_num : 2
activation_offset : true
skip_layers : "layer1.1.conv2"
skip_layer_types:"Conv2d"
do_fusion: true
skip_fusion_layers : "layer1.1.conv2"
common_config : {
arq_quantize : {
    channel_wise : true
}
ifmr_quantize : {
    search_range_start : 0.7
    search_range_end : 1.3
    search_step : 0.01
    max_percentile : 0.999999
    min_percentile : 0.999999
}
}

override_layer_types : {
    layer_type : "Linear"
calibration_config : {
arq_quantize : {
    channel_wise : false
}
ifmr_quantize :{
    search_range_start : 0.8
    search_range_end : 1.2
    search_step : 0.02
    max_percentile : 0.999999
    min_percentile : 0.999999
}
}

override_layer_configs : {
    layer_name : "layer1.2.conv2"
calibration_config : {
arq_quantize : {
    channel_wise : true
}
ifmr_quantize : {
    search_range_start : 0.8
    search_range_end : 1.2
    search_step : 0.02
}
}

```
4.9.2 Simplified Quantization Aware Training Configuration File

**Table 4-34** describes the parameters in the `retrain_config_pytorch.proto` template.

**Table 4-34** Parameter description

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTRetrainConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified quantization aware training configuration file of AMCT.</td>
</tr>
<tr>
<td>Optional</td>
<td>uint32</td>
<td>batch_num</td>
<td></td>
<td>Batch count used for quantization.</td>
</tr>
<tr>
<td>Required</td>
<td>RetrainDataQuantConfig</td>
<td>retrain_data_quant_config</td>
<td>Activation retrain configuration.</td>
<td></td>
</tr>
<tr>
<td>Required</td>
<td>RetrainWeightQuantConfig</td>
<td>retrain_weight_quant_config</td>
<td>Weight retrain configuration.</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>string</td>
<td>skip_layers</td>
<td></td>
<td>Layers to skip during quantization.</td>
</tr>
<tr>
<td>Repeated</td>
<td>RetrainOverrideLayer</td>
<td>override_layer_configs</td>
<td>Layer to override.</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>string</td>
<td>skip_layer_types</td>
<td></td>
<td>Types of layers to skip during quantization.</td>
</tr>
<tr>
<td>Repeated</td>
<td>RetrainOverrideLayerType</td>
<td>override_layer_types</td>
<td>Types of layers to override.</td>
<td></td>
</tr>
<tr>
<td>RetrainDataQuantConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Activation retrain configuration.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------</td>
<td>------------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ULQuantize</td>
<td>-</td>
<td>ULQuantize</td>
<td>ulq_quantize</td>
<td>ULQ algorithm configuration.</td>
</tr>
<tr>
<td>Optioanl</td>
<td>ClipMaxMin</td>
<td>clip_max_min</td>
<td>Fixed_min</td>
<td>Initial upper and lower limits. If it is not specified, IFMR is used for initialization.</td>
</tr>
<tr>
<td>Optioanl</td>
<td>bool</td>
<td>fixed_min</td>
<td></td>
<td>Whether to fix the lower limit at 0. Set to <strong>true</strong> for ReLU or <strong>false</strong> for other algorithms.</td>
</tr>
<tr>
<td>ClipMaxMin</td>
<td>-</td>
<td>-</td>
<td>clip_max</td>
<td>Initial upper limit.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>float</td>
<td>clip_min</td>
<td>Initial lower limit.</td>
</tr>
<tr>
<td>RetrainWeightQuantConfig</td>
<td>-</td>
<td>-</td>
<td>arq_retrain</td>
<td>Weight quantization algorithm. Only ARQ is supported in the current version.</td>
</tr>
<tr>
<td>ARQRetrain</td>
<td>-</td>
<td>ARQRetrain</td>
<td>arq_retrain</td>
<td>Weight quantization algorithm. Only ARQ is supported in the current version.</td>
</tr>
<tr>
<td>Optioanl</td>
<td>Channel-wise</td>
<td>channel_wise</td>
<td></td>
<td>Channel-wise ARQ enable.</td>
</tr>
<tr>
<td>RetrainOverrideLayer</td>
<td>-</td>
<td>-</td>
<td>layer_name</td>
<td>Layer overriding configuration.</td>
</tr>
</tbody>
</table>

**Message:** Required/Optional, **Type:** ULQuantize, **Field:** ulq_quantize

Activation quantization algorithm. Only ULQ is supported in the current version.

**ULQuantize:**

- **Type:** ULQuantize
- **Field:** ulq_quantize

ULQ algorithm configuration.

**ClipMaxMin:**

- **Required:** float
- **Field:** clip_max

Initial upper limit.

- **Required:** float
- **Field:** clip_min

Initial lower limit.

**RetrainWeightQuantConfig:**

- **Required:** string
- **Field:** layer_name

Layer name.
<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RetrainOverrideLayerType</td>
<td>Required</td>
<td>RetrainDataQuantConfig</td>
<td>retrain_data_quant_config</td>
<td>Activation quantization configuration to apply.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>RetrainWeightQuantConfig</td>
<td>retrain_weight_quant_config</td>
<td>Weight quantization configuration to apply.</td>
</tr>
</tbody>
</table>

The following is an example simplified quantization aware training configuration file *(quant.cfg).*

```python
# Global quantization parameters
retrain_data_quant_config:
  ulq_quantize:
    clip_max_min:
      clip_max: 6.0
      clip_min: -6.0
    fixed_min: true

skip_layers: "conv2"
skip_layer_types: "Conv2d"

override_layer_types:
  layer_type: "Linear"
  retrain_weight_quant_config: 
    arq_retrain:
      channel_wise: false

override_layer_configs:
  layer_name: "conv2"
  retrain_data_quant_config:
    ulq_quantize:
      clip_max_min:
```

CANN
Development Auxiliary Tool Guide (Training)  4 AMCT Instructions (PyTorch)

Issue 01 (2021-07-01) Copyright © Huawei Technologies Co., Ltd. 173
```python
clip_max: 3.0
clip_min: -3.0

retrain_weight_quant_config:
  arq_retrain:
    channel_wise: true
```

### 4.9.3 Quantization Factor Record File

#### Prototype

The quantization factor record file is a serialized data structure file based on Protobuf. The corresponding Protobuf prototype is defined as follows (you can also find it in `/amct_pytorch/proto/scale_offset_record_pytorch.proto` under the AMCT installation path):

```protobuf
syntax = "proto2";

message SingleLayerRecord {
  optional float scale_d = 1;
  optional int32 offset_d = 2;
  repeated float scale_w = 3;
  repeated int32 offset_w = 4;
  repeated uint32 shift_bit = 5;
  optional bool skip_fusion = 6 [default = true];
}

message MapFiledEntry {
  optional string key = 1;
  optional SingleLayerRecord value = 2;
}

message ScaleOffsetRecord {
  repeated MapFiledEntry record = 1;
}
```

The parameters are described as follows.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleLayerRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization factors for quantization.</td>
</tr>
<tr>
<td>Optional</td>
<td>float</td>
<td>scale_d</td>
<td></td>
<td>Scale factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td>Optional</td>
<td>int32</td>
<td>offset_d</td>
<td></td>
<td>Offset factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td>Message</td>
<td>Required / Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------</td>
<td>-------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>Repeat</td>
<td>float</td>
<td>scale_w</td>
<td>Scale factor for weight quantization. Scalar mode (quantizing the weight of the current layer in a unified manner) and vector mode (quantizing the weight of the current layer in channel-wise mode) are supported. Only Conv2d layers support channel-wise quantization.</td>
<td></td>
</tr>
<tr>
<td>Repeat</td>
<td>int32</td>
<td>offset_w</td>
<td>Offset factor for weight quantization. Similar to scale_w, it also supports the scalar and vector quantization modes and the dimension configuration must be the same as that of scale_w. Currently weight quantization with offset is not supported, and offset_w must be 0.</td>
<td></td>
</tr>
<tr>
<td>Repeat</td>
<td>uint32</td>
<td>shift_bit</td>
<td>Shift factor.</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>bool</td>
<td>skip_fusion</td>
<td>Whether to skip Conv+BN fusion at the current layer. Defaults to false, indicating that the preceding fusion type is performed.</td>
<td></td>
</tr>
<tr>
<td>ScaleOffsetRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Map structure. To ensure compatibility, the discrete map structure is used.</td>
</tr>
<tr>
<td>Repeat</td>
<td>MapFiledEntry</td>
<td>record</td>
<td>Each records a quantization factor of a quantization layer and consists of two members: • <strong>key</strong>: layer name. • <strong>value</strong>: quantization factor defined by <strong>SingleLayerRecord</strong>.</td>
<td></td>
</tr>
<tr>
<td>MapFiledEntry</td>
<td>Optional</td>
<td>string</td>
<td>key</td>
<td>Layer name.</td>
</tr>
<tr>
<td>Optional</td>
<td>SingleLayerRecord</td>
<td>value</td>
<td>Quantization factor configuration.</td>
<td></td>
</tr>
</tbody>
</table>
Beware that the Protobuf protocol does not report an error for repeated settings of optional fields. Instead, the most recent settings are used.

For general quantization layers, a range of parameters need to be configured, including `scale_d`, `offset_d`, `scale_w`, `offset_w`, and `shift_bit`.

```
  record {
    key: "conv1"
    value {
      scale_d: 0.0798481479
      offset_d: 1
      scale_w: 0.00297622895
      offset_w: 0
      shift_bit: 1
      skip_fusion: true
    }
  }

  record {
    key: "layer1.0.conv1"
    value {
      scale_d: 0.00392156886
      offset_d: -128
      scale_w: 0.00106807391
      scale_w: 0.00104224426
      scale_w: 0.0010603976
      offset_w: 0
      offset_w: 0
      offset_w: 0
      offset_w: 0
      shift_bit: 1
      shift_bit: 1
      shift_bit: 1
      shift_bit: 1
    }
  }
```

### Quantization Factors

The scale (a floating-point number) and offset quantization factors need to be provided for data and weight quantization. AMCT uses a unified quantization data structure. See the following expression.

\[
data_{\text{int8}} = \text{clip}_{\text{int8}}\left(\text{round}\left(\frac{\text{data}_{\text{float}}}{\text{scale}}\right) + \text{offset}\right)
\]

The value ranges are as follows:

- \(\text{scale} \in \left[\text{FLT\_EPSILON}, \frac{1}{\text{FLT\_EPSILON}}\right], \text{FLT\_EPSILON} \approx 1.1920929 \times 10^{-7}\)
- \(\text{offset} \in [-128, 127]\)

Quantization algorithms are classified into symmetric quantization algorithms and asymmetric quantization algorithms.

1. **Symmetric quantization algorithm:**

   The original high-precision data and quantized int8 data are converted into \(\text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int8}}\), where \(\text{scale}\) is a float32. To indicate positive and negative numbers, the signed int8 data type is used for \(\text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int8}}\). The following describes how to convert the original data into the int8 format. \(\text{round}\) is a rounding function. The value to be determined by the quantization algorithm is the constant \(\text{scale}\).
Quantization of the weights and activations may be summarized as a process of searching for a scale. Because $data_{int8}$ is a signed number, to ensure symmetry of the ranges represented by positive and negative values, an absolute value operation is first performed on all data, so that the range of the to-be-quantized data is changed to $data_{int8}$, and then scale is determined. The range of positive int8 values is $[0, 127]$. Therefore, scale can be computed as follows:

$$scale = \frac{data_{max}}{127}$$

Therefore, the range of the int8 values is $[-128 \times scale, 127 \times scale]$. Data beyond the range $[-128 \times scale, 127 \times scale]$ is saturated to a boundary value, and then the quantization operation shown in the formula is performed.

2. Asymmetric quantization algorithm:

The difference from symmetric quantization algorithms lies in the data conversion mode. The scale and offset constants also need to be determined.

$$data_{float} = scale \times (data_{uint8} + offset)$$

The uint8 data is converted through calculation based on the original high-accuracy data, which is shown in the following formula:

$$data_{uint8} = round\left( \frac{data_{float}}{scale} - offset \right)$$

scale is an fp32, $data_{uint8}$ is an unsigned int8, and offset is an int8. The data range is $[scale \times offset, scale \times (255 + offset)]$. If a value range of the to-be-quantized data is $[data_{min}, data_{max}]$, scale and offset are computed as follows:

$$scale = \frac{data_{max} - data_{min}}{255}, \quad offset = \frac{data_{max} - data_{min}}{255}$$

Unified quantization data format: By performing simple data conversion of the asymmetric quantization algorithm formula, the quantized data and the symmetric quantization algorithm are of the same type, that is, int. The specific conversion process is as follows:

int8 quantization is used as an example for description. The input original floating-point data is $data_{float}$, the original quantized fixed-point number is $data_{float}$, the quantization scale is scale, and the quantization offset is $data_{float}$ (the algorithm requires zero crossing to avoid too much accuracy drop). The calculation principle of quantization is as follows:

$$data_{int8} = scale \times (data_{uint8} + offset) = scale \times (data_{int8} + offset + 128) = scale \times (data_{int8} - offset)$$

Where,

$$data_{int8} = data_{int8} - 128 \in [-128, 127], offset = -(offset + 128) \in [-128, 127]$$

Through the foregoing conversion, the data may also be converted into the
int8 format. After scale and the converted offset are determined, the int8 data converted from the original floating-point data is as follows:

\[
data_{\text{int8}} = \text{clip}\left\lfloor \text{round}\left(\frac{data_{\text{float}}}{\text{scale}}\right) + \text{offset}\right\rfloor\text{.}
\]

### 4.9.4 Python 3.7.5 Installation on Ubuntu

**Step 1** Check that the Python 3.7.5 development environment is available.

Run the `python3.7.5 --version`, `python3.7 --version`, `pip3.7.5 --version`, and `pip3.7 --version` commands to check whether the environment is available. If the following information is displayed, the environment is available. Otherwise, go to the next step.

```
Python 3.7.5
pip 19.2.3 from /usr/local/python3.7.5/lib/python3.7/site-packages/pip (python 3.7)
```

**Step 2** Install the Python 3.7.5 dependencies.

```
sudo apt-get install -y make zlib1g zlib1g-dev build-essential libbz2-dev libsqlite3-dev libssl-dev libxml2-dev libffi-dev openssl python3-tk
```

libsqlite3-dev must be installed before the Python installation. If the Python 3.7.5 environment has been installed in the user’s OS before the libsqlite3-dev installation, you need to rebuild the Python environment. If python3-tk fails to be installed, see 4.8.1 An Error Message Is Displayed During python3-tk Installation.

**Step 3** Install Python 3.7.5.

1. Run the `wget` command to download the source package of Python 3.7.5 to any directory on the server where AMCT is located:

   ```
   wget https://www.python.org/ftp/python/3.7.5/Python-3.7.5.tgz
   ```

2. Go to the download directory and decompress the source package:

   ```
tar -zxvf Python-3.7.5.tgz
   ```

3. Go to the new folder and run the following configuration, build, and installation commands:

   ```
cd Python-3.7.5
./configure --prefix=/usr/local/python3.7.5 --enable-loadable-sqlite-extensions --enable-shared
make
sudo make install
```

   --prefix specifies the Python installation path. You can modify it as required. --enable-shared is used to build the libpython3.7m.so.1.0 dynamic library. --enable-loadable-sqlite-extensions is used to load the sqlite-devel dependency.

   This document uses --prefix=/usr/local/python3.7.5 as an example. After the configuration, build, and installation commands are executed, the installation package is output to the /usr/local/python3.7.5 directory, and the libpython3.7m.so.1.0 dynamic library is output to the /usr/local/python3.7.5/lib/libpython3.7m.so.1.0 directory.

4. Set the soft links:

   ```
sudo ln -s /usr/local/python3.7.5/bin/python3 /usr/local/python3.7.5/bin/python3.7.5
   sudo ln -s /usr/local/python3.7.5/bin/pip3 /usr/local/python3.7.5/bin/pip3.7.5
   ```

5. Set the Python 3.7.5 environment variables.

   a. If Python is installed by the root user:
AMCT is installed by the root user. Run the following commands in the current terminal window to set environment variables:

```
# Set the Python 3.7.5 library path.
export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
# If multiple Python 3 versions exist in the user environment, specify Python 3.7.5.
export PATH=/usr/local/python3.7.5/bin:$PATH
```

**NOTICE**

If the running user is the root user, it is not advised to modify the `.bashrc` file. Otherwise, the Python tools provided by other systems may be unavailable. If you want to use the default tool, open another terminal window.

b. If Python is installed by a non-root user:

AMCT is also installed by the non-root user. Run the `vi ~/.bashrc` command in any directory to open the `.bashrc` file, and append the following lines to the file:

```
# Set the Python 3.7.5 library path.
export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
# If multiple Python 3 versions exist in the user environment, specify Python 3.7.5.
export PATH=/usr/local/python3.7.5/bin:$PATH
```

Run the `:wq!` command to save the file and exit. Run the `source ~/.bashrc` command for the modification to take effect immediately.

**Step 4** After the installation is complete, run the following commands to check the installed version. If the required version information is displayed, the installation is successful.

```
python3.7.5 --version
pip3.7.5 --version
python3.7 --version
pip3.7 --version
```

----End
5 AMCT Instructions (TensorFlow)

5.1 Introduction
5.2 AMCT Installation
5.3 Post-training Quantization
5.4 Quantization Aware Training
5.5 Accuracy-oriented Automatic Quantization Rollback
5.6 Tensor Decomposition
5.7 AMCT Update
5.8 AMCT Uninstallation
5.9 API Description
5.10 FAQs
5.11 Appendixes

5.1 Introduction

5.1.1 Overview

This document describes how to quantize a TensorFlow model using Ascend Model Compression Toolkit (AMCT). In the quantization process, the precision of model weights and activations is reduced to make the model more compact, improving the compute efficiency while lowering the transfer latency.

AMCT is a TensorFlow-based Python toolkit that implements layer fusion and 8-bit quantization of data and weights in neural networks. This toolkit decouples model quantization from model conversion. It implements independent quantization of quantization-capable layers in a model and saves the quantized model to a .pb file. The obtained accuracy simulation model can run on CPU or GPU to complete accuracy simulation. The obtained deployable model can run on the Ascend AI Processor with improved inference performance. Currently, AMCT supports only the fp32 data type. The quantized model can serve for both accuracy simulation and inference deployment. This tool has the following advantages:
Lightweight: You only need to install the tool.
Easy-to-use APIs: You can complete quantization using APIs based on the TensorFlow inference script.
Hardware compatibility: The quantized model can be converted to an offline model by using the ATC tool to implement 8-bit inference on Ascend AI Processor.
Configurable quantization: You can modify the quantization configuration file and adjust the quantization strategy to obtain the optimal quantization result.

#EN-US_TOPIC_0000001095145848/en-us_topic_0240188001_fig169831649143814 shows the application scenarios. Currently, AMCT runs only on Ubuntu 18.04. For the architecture mapping details, see Setting Up Environment. Before you run inference on the Ascend AI Processor with the model quantized by this tool, you need to use Ascend Tensor Compiler (ATC) to convert the quantized model to an offline model adapted to the Ascend AI Processor.

5.1.2 Features

5.1.2.1 Post-training Quantization and Quantization Aware Training

5.1.2.1.1 Terminology

There are two forms of quantization: post-training quantization and quantization aware training.

The foregoing two quantization forms are classified into weight quantization and activation quantization according to the quantization object, and are classified into uniform quantization and non-uniform quantization (only uniform quantization is supported) according to whether weights are compressed. Quantization aware training allows only uniform quantization.

As used in this document, the following terms have the meanings specified below.

Post-training Quantization

Post-training quantization refers to quantizing the weights of an already-trained model from float32 to int8 and calibrating and quantizing the activations by using a small calibration dataset. For details about the quantization workflow, see 5.3 Post-training Quantization. It is not supported to run a post-training quantization on more than one GPU.

- Calibration dataset
  During the calibration process, the algorithm uses each piece of data in the calibration dataset as input, accumulates the input data of each layer (or operator) that needs to be quantized, and uses the accumulated data as input data of the quantization algorithm to determine the quantization factors. Because the quantization factors and the accuracy of the quantized model are closely related to the selection of the calibration dataset, you are advised to calibrate the model using a subset of images from the validation dataset.

- Activation quantization
Activation quantization is to collect statistics on the input data of each layer (or operator) to be quantized, to find the optimal pair of scale and offset per layer (or operator). For details, see 5.11.4 Quantization Factor Record File.

Activations are the intermediate results of model inference computation. The value ranges are input-specific. Therefore, a group of reference inputs (a calibration dataset) needs to be used as incentives to record the input data of each layer (or operator) to be quantized for searching for quantization factors (scale and offset). During data calibration, extra memory (video memory/RAM) is needed to store the input data used to determine the quantization factors. Therefore, the video memory/RAM usage is higher than that required for performing inference only. The size of the extra memory is positively correlated with $\text{batch}_\text{size} \times \text{batch}_\text{num}$ during calibration.

- **Weight quantization**

  After model training, the weights and the value ranges are determined. Therefore, quantization may be directly performed based on the value range of each weight.

Based on whether the weight data is compressed, quantization is classified into uniform quantization and non-uniform quantization. However, only uniform quantization is supported.

**Uniform quantization**: The quantized data is evenly distributed in a numerical space. For example, int8 quantization uses 8-bit int8 data to represent 32-bit fp32 data, and converts an fp32 operation (multiply-add operation) into an int8 operation, accelerating computing with reduced model size. In uniform int8 quantization, the quantized data is evenly distributed in the value range $[-128, +127]$ of int8. For details about the quantization workflow, see 5.3.1 Uniform Quantization.

If the accuracy drops significantly after uniform quantization, you need to retrain the model by referring to Quantization Aware Training or 5.1.2.2 Accuracy-oriented Automatic Quantization Rollback. The layers that support post-training quantization are listed as follows.

### Table 5-1 Layers that support post-training quantization and restrictions

<table>
<thead>
<tr>
<th>Layer</th>
<th>Restriction</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatMul</td>
<td>transpose_a=False, transpose_b=False, adjoint_a=False, adjoint_b=False</td>
<td>-</td>
</tr>
<tr>
<td>Conv2D</td>
<td>-</td>
<td>The input source of weight does not contain nodes that can be dynamically changed, such as placeholder, and the node type of weight can only be const.</td>
</tr>
<tr>
<td>DepthwiseConv2dNative</td>
<td>dilation = 1</td>
<td></td>
</tr>
<tr>
<td>Conv2DBackpropInput</td>
<td>dilation = 1</td>
<td></td>
</tr>
<tr>
<td>AvgPool</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
Quantization Aware Training

Quantization aware training introduces quantization in the forward pass of the training process, allowing for higher accuracy.

Quantization aware training is time consuming and data hungry. For details about the quantization workflow, see 5.4 Quantization Aware Training.

- **Training dataset**
  Dataset of the already-trained network.

- **Activation quantization**
  Activation quantization refers to iterative training of the upper clip limit and lower clip limit, which are used to calculate the current scale and offset. The activation is the intermediate result of model inference and calculation. The ULQ retrain algorithm is used to continuously optimize the two factors during the quantization aware training process to obtain the optimal factors.

- **Weight quantization**
  Weight quantization means to optimize the quantization parameters of weights during the quantization aware training process to obtain the optimal parameters.

The layers that support quantization aware training are listed as follows.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatMul</td>
<td>transpose_a=False, transpose_b=False, adjoint_a=False, adjoint_b=False</td>
</tr>
<tr>
<td>Conv2D</td>
<td>-</td>
</tr>
<tr>
<td>DepthwiseConv2dNative</td>
<td>dilation = 1</td>
</tr>
<tr>
<td>Conv2DBackpropInput</td>
<td>dilation = 1</td>
</tr>
<tr>
<td>AvgPool</td>
<td>-</td>
</tr>
</tbody>
</table>

5.1.2.1.2 Principles

Figure 5-1 shows the AMCT principles. The operations in blue are implemented by users, and the operations in gray are implemented by using AMCT API calls. You can import the library to the original TensorFlow network inference code and call the APIs at specific locations to implement the quantization function. The tool can be used in the following scenarios:

- **Post-training Quantization**
  - **Scenario 1**
    i. Construct an original TensorFlow model and then call the 5.9.2.1 create_quant_config API to generate a quantization configuration file.
    ii. Call the 5.9.2.2 quantize_model API to optimize the original TensorFlow model based on the quantization configuration file. The
optimized model contains the quantization algorithm. Run inference with the optimized model in the TensorFlow environment based on the image dataset and calibration dataset preset in AMCT.

The image dataset is used to test the accuracy of the quantized data in the TensorFlow environment. The calibration dataset is used to generate quantization factors to ensure accuracy.

iii. Call the **5.9.2.3 save_model** API to save the quantized model which can serve for both accuracy simulation in the TensorFlow environment and deployment on the Ascend AI Processor.

For details about APIs, see **5.9 API Description**.

- **Scenario 2**

Instead of using the APIs in scenario 1, if you have generated a quantized model based on your own quantization factors and original TensorFlow model, call the **5.9.2.4 convert_model** API to complete the quantization. **5.3.1.3 Quantization Example Using the convert_model API** gives a quantization example in this scenario.

- **Quantization Aware Training**

a. Construct a training graph (that is, set **is_training** to **True** for BN), and then call the **5.9.3.1 create_quant_retrain_config** API to generate a quantization configuration file (corresponding to 1 in Figure 5-1).

b. Call the **5.9.3.2 create_quant_retrain_model** API to modify the graph before quantization based on the quantization configuration file (corresponding to 2 in Figure 5-1).

c. Call the adaptive learning rate optimizer (RMSPropOptimizer) to create a backpropagation computational graph (perform this step after b).

```python
optimizer = tf.compat.v1.train.RMSPropOptimizer(ARGS.learning_rate, momentum=ARGS.momentum)
train_op = optimizer.minimize(loss)
```

d. Create a session to train the model, and save the trained parameters as a checkpoint file (corresponding to 3 and 4 in Figure 5-1).

```python
with tf.Session() as sess:
sess.run(tf.compat.v1.global_variables_initializer())
sess.run(outputs)
# Save the trained parameters as a checkpoint file.
saver_save.save(sess, retrain_ckpt, global_step=0)
```

e. Construct an inference graph (that is, set **is_training** to **False** for the BN), and then call the **5.9.3.2 create_quant_retrain_model** API to modify the graph before quantization based on the quantization configuration file (corresponding to 5 in Figure 5-1).

f. Create a session to restore training parameters, perform inference to obtain the quantization output node (retrain_ops[-1]), write quantization factors into the record file, and save the model as a PB model (corresponding to 6 and 7 in Figure 5-1).

```python
variables_to_restore = tf.compat.v1.global_variables()
saver_restore = tf.compat.v1.train.Saver(variables_to_restore)
with tf.Session() as sess:
sess.run(tf.compat.v1.global_variables_initializer())
# Restore training parameters.
saver_restore.restore(sess, retrain_ckpt)
# Write quantization factors into the record file.
sess.run(retrain_ops[-1])
# Save the model as a PB model.
constant_graph = tf.compat.v1.graph_util.convert_variables_to_constants(
sess, eval_graph.as_graph_def(), [output.name[:-2] for output in outputs])
```
g. Call the **5.9.3.3 save_quant_retrain_model** API to export the quantized model which can server for both accuracy simulation in the TensorFlow environment and deployment on the Ascend AI Processor based on the quantization factor record file (corresponding to 8 in **Figure 5-1**).

**Figure 5-1 Tool principles**

---

5.1.2.2 Accuracy-oriented Automatic Quantization Rollback

Accuracy-oriented automatic quantization rollback is a technique introduced to ensure model accuracy, which automatically searches for the model quantization configuration and executes the post-training quantization process without sacrificing accuracy.

Accuracy-oriented automatic quantization rollback is similar to the post-training quantization process. However, you do not need to manually tune the quantization configuration file, which greatly simplifies the optimization workload and improves the quantization efficiency. **Figure 5-2** shows the working principles.

**Figure 5-2 Working principles**

---

The workflow is described as follows:

1. Generate a quantization configuration file by using the **5.9.2.1 create_quant_config** call, and then perform accuracy-oriented automatic
quantization rollback by using the 5.9.2.5 accuracy_based_auto_calibration call.

2. Pass the evaluator instance to the 5.9.2.5 accuracy_based_auto_calibration call to analyze the accuracy of the original model.

In this process, the quantization strategy module in 5.9.2.5 accuracy_based_auto_calibration is called to output the initialized quantization configuration file. The file records all layers that support quantization.

3. Run quantization on the model based on the initial quantization configuration file (generated by the 5.9.2.1 create_quant_config call in 1) to obtain the accuracy of the fake-quantized model.

4. Compare the accuracy results of both models. If the fake-quantized model's degradation in accuracy is within acceptable limits, the quantized model is output. Otherwise, perform quantization rollback.
   a. Run inference on the original TensorFlow model and dump the input activations of each layer.
   b. Use the calibrated quantization factors to construct single-operator networks of quantization layers. Then, use the buffered activations to calculate the cosine similarity between the output data of each fake-quantized single-operator network and that of the original TensorFlow equivalent.
   c. Pass the cosine similarity list to the quantization strategy module in 5.9.2.5 accuracy_based_auto_calibration. The strategy module outputs a new quantization configuration file after certain layers are rolled back based on the initial quantization configuration file generated in 2.
   d. Run post-training quantization based on the new quantization configuration file to obtain a new fake-quantized model.
   e. Analyze the accuracy of the new fake-quantized model by a call to the evaluator module in 5.9.2.5 accuracy_based_auto_calibration.
      - If the model accuracy is acceptable, output a fake-quantized model and a deployable model.
      - If the model accuracy is unacceptable, the layer with the worst cosine similarity is rolled back, and go back to 4.c to output a new quantization configuration.

*Figure 5-3* shows the rollback workflow.
5.1.2.3 Tensor Decomposition

Overview

Tensor decomposition converts a convolutional kernel into a stack of two smaller convolutional kernels to reduce the inference overhead. If the user model involves many convolution computations and most of the convolutional kernels have shapes larger than (64, 64, 3, 3), tensor decomposition is recommended. Otherwise, skip this step and proceed to quantization. Currently, tensor decomposition is supported under the following conditions:

- group = 1, dilation = (1,1), stride < 3
- kernel_h = kernel_w, kernel_h > 2
Only when the original TensorFlow model has the Conv2D layer and the layer meets the preceding conditions, the Conv2D layer can be decomposed into two smaller Conv2D layers. Then, you can use AMCT to convert the original TensorFlow model into a quantizable model that can be deployed on the Ascend AI Processor for better inference performance.

**Decomposition Principles**

*Figure 5-4* shows the decomposition principle. Determine whether to decompose the original model as needed. For the tensor decomposition details, see [5.6 Tensor Decomposition](#).

*Figure 5-4* Tensor decomposition principles

The procedure is as follows:


2. Call [5.9.4.2 decompose_graph](#) to decompose the graph in the training code based on the TensorFlow training-mode graph and the output graph structure changes output by the [5.9.4.1 auto_decomposition](#) call.

3. Fine-tune the decomposed model. Quantize the fine-tuned model by referring to [5.3 Post-training Quantization](#) or [5.4 Quantization Aware Training](#).

*Figure 5-5* shows the resnet_v2_50 model before and after decomposition.
### 5.1.2.4 QAT Model to Ascend Model Conversion

#### Overview

An already-quantized original TensorFlow model is referred to as a QAT model. Prior to generating an offline model adapted to the Ascend AI Processor with ATC, you need to use AMCT’s 5.9.5.1 `convert_qat_model` API to convert the QAT model into an Ascend representation, which means that ATC does not support direct conversion from a QAT model to an offline model. Note the following restrictions on QAT model to Ascend model conversion:

- The source QAT model must have FakeQuant layers, including FakeQuantWithMinMaxVars and FakeQuantWithMinMaxVarsPerChannel (weights only).
- Only the Conv2D, MatMul, DethwiseConv2dNative, Conv2dBackproInput, and AvgPool layers can match `fake_quant` nodes, which means only these layers are converted. Table 5-1 shows the layer restrictions.

---

**Figure 5-5** Model before and after decomposition

![Diagram of model before and after decomposition](image)
Principles

Figure 5-6 shows the conversion principles. In the figure, the blue part is implemented by the user, and the gray part is implemented by the call to AMCT's 5.9.5.1 convert_qat_model API. The user needs to import the dependence library to the TensorFlow QAT model inference code and inserts the corresponding API call to implement the conversion. 5.1.2.2 Accuracy-oriented Automatic Quantization Rollback does not apply to QAT models.

Figure 5-6 Converting a QAT Model into an Ascend Quantized Model

QAT model to Ascend model conversion

5.1.2.5 Fusion Support

Currently, this tool mainly implements the following forms of layer fusion:

- Conv+BN fusion: Before AMCT quantization, the "Conv2D+BatchNorm" composite in the model is fused into "Conv+BN." The BatchNorm layer is removed.
- Depthwise_Conv+BN fusion: Before AMCT quantization, the "DepthwiseConv2dNative+BatchNorm" composite in the model is fused into "Depthwise_Conv+BN." The BatchNorm layer is removed.
- OP+(BiasAdd)+Mul fusion: Before AMCT quantization, the "Conv2D/MatMul/DepthwiseConv2dNative/Conv2DBackpropInput+Mul" and "Conv2D/MatMul/DepthwiseConv2d/Conv2DBackpropInput+BiasAdd+Mul" composites in the model are fused into "OP+(BiasAdd)+Mul." After the fusion, the Mul layer is removed.

In this scenario, the other input of Mul must be of the Const type with an empty shape.
- Group_conv+BN fusion: If the "Split+Multi-Conv2D+ConcatV2 (or Concat, with concatenation performed along the C dimension)" composite is used in the model to indicate Group_conv, the "Group_conv+BatchNorm" composite is fused before AMCT quantization. The BatchNorm layer is removed.

BN fusion applies to the following operators: FusedBatchNorm, FusedBatchNormV2, and FusedBatchNormV3.
- MatMul+BN fusion:
  - The output shape of the Reshape operator must be the same as that of the MatMul operator, as shown in Figure 5-7. The MatMul operator must meet any of the following conditions:
    - No biasadd, NHWC format, dynamic shape
    - No biasadd, NCHW format, dynamic shape
    - No biasadd, NHWC format, static shape
    - No biasadd, NCHW format, static shape
- With biasadd, NHWC format, dynamic shape
- With biasadd, NCHW format, dynamic shape
- With biasadd, NHWC format, static shape
- With biasadd, NCHW format, static shape

Figure 5-7 MatMul+BN fusion diagram

- BN fusion applies to the following operators: FusedBatchNorm, FusedBatchNormV2, and FusedBatchNormV3.
- Small BN operators are fused into FusedBatchNormV3 on the condition that the small BN operators take 4D inputs. This fusion is supported only by post-training quantization.

AMCT analyzes the composite of the small BN operators generated by `tf.keras.layers.BatchNormalization`, and replaces the small BN operators with larger BN composite on the following conditions:
- On `tf.keras.layers.BatchNormalization` with `fused=False` and `training=False`, the network structures before and after fusion are as follows.
- On `tf.keras.layers.BatchNormalization` with `fused=False`, `center=False`, and `training=False`, the network structures before and after fusion are as follows.

- On `tf.keras.layers.BatchNormalization` with `fused=False`, `scale=False`, and `training=False`, the network structures before and after fusion are as follows.
On `tf.keras.layers.BatchNormalization` with `fused=False`, `scale=False`, `center=False`, and `training=False`, the network structures before and after fusion are as follows.

5.1.3 Tool Workflow

Figure 5-8 shows the tool workflow.
Figure 5-8 Tool workflow

Table 5-3 Major actions in the tool workflow

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package preparation</td>
<td>Obtain the tool package by referring to <a href="#">5.2.1 Package Preparation</a></td>
</tr>
<tr>
<td>Pre-installation actions</td>
<td>Before AMCT installation, create an installation user, check the system environment, install dependencies, and upload the AMCT package. For details, see <a href="#">5.2.2 Pre-installation Actions</a>.</td>
</tr>
<tr>
<td>Installation</td>
<td>Install the TensorFlow version AMCT by referring to <a href="#">5.2 AMCT Installation</a>.</td>
</tr>
<tr>
<td>(Optional) Script creation with AMCT API calls</td>
<td>If you need to quantize your network model instead of the sample model provided in this instruction, you need to modify the script for adaptation before quantization.</td>
</tr>
</tbody>
</table>
### 5.2 AMCT Installation

#### 5.2.1 Package Preparation

Currently, AMCT runs only on Ubuntu 18.04 (x86_64) servers. Before installation, click [here](#) to obtain the AMCT package `Ascend-cann-amct_{software version}_ubuntu18.04-x86_64.tar.gz`.  

Before installation, obtain the AMCT package. AMCT runs on Ubuntu 18.04 (x86_64) or EulerOS (AArch64) servers. Select a required software package.

- **Ubuntu 18.04 (x86_64) server:**
  - `Ascend-amct_{software version}-ubuntu18.04.x86_64.tar.gz`
- **EulerOS (AArch64) server:**
  - `Ascend-amct_{software version}-euleros2.9.aarch64.tar.gz`

_{software version} indicates the version number.

#### 5.2.2 Pre-installation Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tensor decomposition</td>
<td>If the user model involves many convolution computations and most of the convolutional kernels have shapes larger than (64, 64, 3, 3), tensor decomposition is recommended. Otherwise, skip this step and proceed to quantization. For details about the tensor decomposition procedure, see 5.6 Tensor Decomposition.</td>
</tr>
<tr>
<td>Quantization</td>
<td>Run the provided quantization script to quantize an original network with the prepared dataset. There are two forms of quantization: post-training quantization and quantization aware training. For details, see 5.3 Post-training Quantization and 5.4 Quantization Aware Training. Post-training quantization can be further classified into uniform quantization and non-uniform quantization according to whether the weight data is compressed. This version supports only uniform quantization.</td>
</tr>
<tr>
<td>Accuracy-oriented automatic quantization rollback</td>
<td>Check the accuracy of the quantized model. If the accuracy of the quantized model is not satisfactory, perform 5.5 Accuracy-oriented Automatic Quantization Rollback or 5.4 Quantization Aware Training.</td>
</tr>
<tr>
<td>(Optional) Model conversion using ATC</td>
<td>You can convert the quantized deployable model to an offline model supported by the Ascend AI Processor by using ATC, and then perform subsequent inference.</td>
</tr>
</tbody>
</table>
5.2.2.1 Ubuntu (x86)

Preparing the AMCT User

Any user (root or non-root) is allowed to install AMCT. This section uses a non-root user as an example.

- To install AMCT as the root user, skip this section.
- To install AMCT as an existing non-root user, ensure that the user has the read, write, and execute permissions on the $HOME directory.
- To install AMCT as a new non-root user, perform the following steps as the root user. The following uses this scenario as an example.
  a. Create an AMCT installation user and set the $HOME directory for the user:
     ```bash
     useradd -d /home/username -m username
     ```
  b. Set the user password:
     ```bash
     passwd username
     ```

  **NOTE**

  *username* indicates the name of the AMCT installation user. The *umask* value of the user is at least 0027.
  - You can view the *umask* value by running the *umask* command.
  - You can change the *umask* value by running the *umask NewValue* command.

(Optional) Setting the Permission of the AMCT Installation User

Skip this part if you install AMCT as the root user.

Before installing AMCT, you need to download the dependencies, which requires the *sudo* `apt-get` permission. Run the following commands as the root user:

1. Open the `/etc/sudoers` file:
   ```bash
   chmod u+w /etc/sudoers
   vi /etc/sudoers
   ```
2. Add the following content under `# User privilege specification` in the file:
   ```bash
   username ALL=(ALL:ALL)  NOPASSWD:SETENV:/usr/bin/apt-get,/usr/bin/pip, /bin/tar, /bin/mkdir, /bin/sh, /bin/bash, /usr/bin/make, /usr/bin/pip3, /usr/bin/pip3.7, /usr/bin/pip3.7.5, /bin/ln
   ```
   Replace *username* with the name of the non-root user who executes the installation script.

  **NOTE**

  Check if the last line in the `/etc/sudoers` file is `#includedir /etc/sudoers.d`. If no, add it manually.

3. Run the `:wq!` command to save the file.
4. Remove the write permission on the `/etc/sudoers` file:
   ```bash
   chmod u-w /etc/sudoers
   ```
Setting Up Environment

**NOTE**

This tool supports Python 3.7.0 to Python 3.7.9. This document uses Python 3.7.5 as an example. The environment variables and installation commands vary according to the actual Python version.

Currently, AMCT runs on Ubuntu 18.04 (x86_64). The following table details the architecture mapping.

**Table 5-4** Ubuntu (x86_64) architecture mapping

<table>
<thead>
<tr>
<th>Item</th>
<th>Version</th>
<th>How to Obtain</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Ubuntu 18.04 (64-bit)</td>
<td>Click <a href="#">here</a> to download an Ubuntu release. The <strong>ubuntu-18.04-server-amd64.iso</strong> server install image is recommended.</td>
<td>-</td>
</tr>
<tr>
<td>Python</td>
<td>3.7.5</td>
<td>See <a href="#">5.11.8 Python 3.7.5 Installation on Ubuntu</a>.</td>
<td>Make sure that the server has Internet access.</td>
</tr>
<tr>
<td>CUDA Toolkit/CUDA Driver</td>
<td>10.0</td>
<td>Obtain required packages for installation. For example, you can obtain the Toolkit package from the following URL, which contains the Driver package. <a href="#">https://developer.nvidia.com/cuda-toolkit-archive</a>.</td>
<td>To perform GPU quantization, the CUDA software must be installed.</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>1.15</td>
<td>See <a href="#">Installing Dependencies</a>.</td>
<td>-</td>
</tr>
<tr>
<td>Numpy</td>
<td>1.16.0+</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Pillow</td>
<td>6.0.0+</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Protobuf</td>
<td>3.11.0+</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Tqdm</td>
<td>4.55.0</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Matplotlib</td>
<td>3.2.0</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
Checking Sources

During dependency installation, you need to make sure that the server of AMCT has Internet access. Run the following command as the root user to check source validity:

```
apt-get update
```

If an error is reported during the command execution, check whether the network connection is normal or replace the source in the `/etc/apt/sources.list` file with a valid one.

Installing Dependencies

Use the AMCT installation user to install software. If the installation user is a non-root user, run the `su - username` command to switch to the non-root user and run the following commands.

### Table 5-5 Dependency list

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Version</th>
<th>Installation Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>3.7.5</td>
<td>See <a href="#">5.11.8 Python 3.7.5 Installation on Ubuntu</a>.</td>
</tr>
</tbody>
</table>
| TensorFlow | 1.15    | Install a CPU or GPU version as needed.  
|            |         | - CPU version  
|            |         |   python3.7.5 -m pip install tensorflow-cpu==1.15 --user  
|            |         | - GPU version  
|            |         |   python3.7.5 -m pip install tensorflow-gpu==1.15 --user |
| NumPy      | 1.16.0+ | pip3.7.5 install numpy==1.16.0 --user |
| Pillow     | 6.0.0+  | pip3.7.5 install pillow==6.0.0 --user |
| Protobuf   | 3.11.0+ | pip3.7.5 install protobuf==3.11.0 --user |
| tqdm       | 4.55.0  | pip3.7.5 install tqdm==4.55.0 --user |
| Matplotlib | 3.2.0   | pip3.7.5 install matplotlib==3.2.0 --user |

Uploading the AMCT Package

Upload the `Ascend-cann-amct_{software version}_ubuntu18.04-x86_64.tar.gz` package to any directory (for example, `S HOME/amct`) on the Linux server as the AMCT installation user.

Decompress the AMCT package:

```
tar -zxvf Ascend-cann-amct-{software version}_ubuntu18.04-x86_64.tar.gz
```

Find the following extracted packages.
<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
</table>
| **amct_caffe/**   | **Caffe AMCT directory** | | ● OS support: Ubuntu 18.04 (x86_64)  
● For details, see AMCT Instructions (Caffe).  
● Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| amct_caffe-  
{version}-py3-none-linux_{arch}.whl | Caffe AMCT package | | |
| amct_caffe_sample.tar.gz | Caffe quantization sample package | | |
| caffe_patch.tar.gz | Caffe patch package | | |
| **amct_tensorflow/** | **TensorFlow AMCT directory** | | ● OS support: Ubuntu 18.04 (x86_64)  
● `amct_tensorflow` and `amct_tensorflow_ascend` cannot exist at the same time.  
● For details, see AMCT Instructions (TensorFlow).  
● Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| amct_tensorflow-  
{version}-py3-none-linux_{arch}.whl | TensorFlow AMCT package | | |
| amct_tensorflow_sample.tar.gz | TensorFlow quantization sample package | | |
| amct_tensorflow_ascend-  
{version}-py3-none-linux_{arch}.whl | TF Adapter AMCT package | | ● OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
● `amct_tensorflow` and `amct_tensorflow_ascend` cannot exist at the same time.  
● For details, see AMCT Instructions (TensorFlow, Ascend).  
● Inference on a quantized model needs to be performed in the Ascend 310 inference environment |
<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>amct_tensorflow_ascend_sample.tar.gz</strong></td>
<td>Package of quantization samples using TF Adapter</td>
<td><strong>installed with the Ascend AI Processor.</strong></td>
<td></td>
</tr>
</tbody>
</table>
| **amct_pytorch/** | PyTorch AMCT directory | | • OS support: Ubuntu 18.04 (x86_64)  
• For details, see *AMCT Instructions (PyTorch)*.  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| **amct_onnx/** | ONNX AMCT directory | | • OS support: Ubuntu 18.04 (x86_64)  
• For details, see *AMCT Instructions (ONNX)*.  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| **amct_mindspore/** | MindSpore AMCT directory | | • OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
• For details, see *AMCT Instructions (MindSpore)*.  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| **amct_ascendcl/** | AscendCL API AMCT directory | | • OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
• For details, see *AMCT Instructions (AscendCL)*. |
<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>amct_acl_sample.tar.gz</td>
<td>Package of quantization samples using AscendCL APIs</td>
<td>● Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor.</td>
</tr>
</tbody>
</table>

{version} indicates the AMCT version number. {os}.{arch} indicates the OS and architecture.

### 5.2.3 Installation

**Step 1** In the directory where the AMCT package is located, run the following command:

```
pip3.7.5 install amct_tensorflow-{version}-py3-none-linux_{arch}.whl --user
```

Replace {version} with the actual AMCT version number, and {arch} with the actual architecture of the installation server. If AMCT installation is performed by the root user and the --target option is included, ensure that the path specified by --target is the path of the current user.

**Step 2** Check the installation. If a message similar to the following is displayed, the installation is successful:

Successfully installed amct-tensorflow-{version}

Find the installed AMCT in the python3.7.5 directory (for example, $HOME/.local/lib/python3.7.5/site-packages).

```
   drwxr-xr-x  5 amct amct   4096 Mar 17 11:50 amct_tensorflow
   drwxr-xr-x  2 amct amct   4096 Mar 17 11:50 amct_tensorflow-{version}.dist-info

amct_tensorflow indicates the AMCT installation path.

----End
```

### 5.3 Post-training Quantization

#### 5.3.1 Uniform Quantization

#### 5.3.1.1 Quantizing an Image Classification Network

#### 5.3.1.1.1 Quantization Preparations

**Prerequisites**

- Model
Upload the TensorFlow model to be quantized to any directory on the Linux server as the AMCT installation user. The following uses the `mobilenet_v2.pb` model available in the sample package as an example.

**NOTE**
If you choose to use your own model, you are advised to perform inference in the TensorFlow environment in advance to test if it can run properly in the TensorFlow environment with expected accuracy.

- **Image dataset**
  After the model is quantized using the AMCT, perform inference with the model to test the model accuracy. Use the dataset that matches the model.
  Upload the dataset matching the model to any directory on the Linux server as the AMCT installation user.

- **Calibration dataset**
  The calibration dataset is used to generate the quantization factors to guarantee the accuracy.
  The process of calculating the quantization factors is referred to as calibration. Perform inference using the quantized network with one or more batches of a subset of images from the validation dataset to complete calibration.
  Upload the calibration dataset file to any directory on the Linux server as the AMCT installation user.

The following uses the MobileNetV2 network in the quantization sample package as an example. The model file and label dictionary in the dataset are automatically downloaded when the quantization script is executed.

### 5.3.1.2 Quantization Example

The following uses the image classification network quantization script `MobileNetV2_sample.py`, the `mobilenet_v2.pb` model file, the `classification.jpg` dataset, and the `calibration` dataset available in the sample package to illustrate how to execute the quantization script.

1. Obtain the quantization script.
   In the directory of `amct_tensorflow_sample.tar.gz`, extract the quantization script from the package:
   ```
tar -zxvf amct_tensorflow_sample.tar.gz

cd sample
   ```
   Find the following extracted directories:
   - `mobilenetv2/`: quantization directory of the image classification network MobileNetV2. This section describes how to execute the quantization script for image classification network.
   - `yolov3/`: YOLOv3 quantization directory. For details, see 5.3.1.2 Quantizing an Object Detection Network.
   - `resnet_v1_50/`: directory of quantization aware training. For details, see 5.4 Quantization Aware Training.
   - `tensor_decompose/`: tensor decomposition directory. For details, see 5.6 Tensor Decomposition.

For details about the directory structure, see 5.11.1 Sample Directory Description.
2. Run the quantization script.

Run the following command in the sample/mobilenetv2 directory to quantize the MobileNetV2 network:

```
python3.7.5 MobileNetV2_sample.py [-h] [--cfg_define CFG_DEFINE]
```

Table 5-7 describes the command-line options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--h</td>
<td>(Optional) Displays help information.</td>
</tr>
<tr>
<td>--cfg_define</td>
<td>(Optional) Sets the directory of the simplified</td>
</tr>
<tr>
<td>CFG_DEFINE</td>
<td>quantization configuration file.</td>
</tr>
</tbody>
</table>

An example is as follows:

```
python3.7.5 MobileNetV2_sample.py
```

If messages similar to the following are displayed, the quantization is successful:

```
INFO - [AMCT]:[save_model]: The model is saved in $HOME/amct/amct_tensorflow/log/results/calibration/MobileNetV2_quantized.pb
```

An example is as follows:

```
INFO - [AMCT]:[save_model]: The model is saved in $HOME/amct/amct_tensorflow/log/results/calibration/MobileNetV2_quantized.pb
```

Origin Model Prediction:

<table>
<thead>
<tr>
<th>category index</th>
<th>category prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>699</td>
<td>0.560</td>
</tr>
</tbody>
</table>

Quantized Model Prediction:

<table>
<thead>
<tr>
<th>category index</th>
<th>category prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>699</td>
<td>0.575</td>
</tr>
</tbody>
</table>

3. View the quantization result.

After the quantization is complete, find the following files generated in the directory of the quantization script:

- **config.json**: configuration file that describes how to quantize each layer in the model. If a quantization configuration file already exists in the directory of the quantization script, the existing quantization configuration file is overwritten by a new one with the same name in a call to **5.9.2.1 create_quant_config**. If not that case, a new quantization configuration file is created.

If the accuracy of model inference drops significantly after quantization, you can modify the **config.json** file by referring to **5.3.2 Quantization Configuration**.

- **amct_log/amct_tensorflow.log**: quantization log file.
- **record.txt**: file that records quantization factors. For details about the prototype definition of the file, see **5.11.4 Quantization Factor Record File**.
- **results/calibration**: quantization result directory, containing the quantized model file and its quantization information file **MobileNetV2_quant.json** (named after the quantized model).

- **MobileNetV2_quant.json**: quantization information file. This file gives the node mapping between the quantized model and the...
original model and is used for accuracy comparison between the quantized model and the original model.

- **MobileNetV2_quantized.pb**: quantized model that can both serve for accuracy simulation in the TensorFlow environment and be deployed on the Ascend AI Processor.

When a model is re-quantized, the existing result files will be overwritten.

---

**NOTICE**

- If a message is displayed during AMCT run time, indicating that the resource allocation fails, fix the error by referring to [5.10.2 AMCT Resource Allocation Failure](#).
- If you want to use your own inference script to perform inference with the quantized model in the TensorFlow environment, import the AMCT package by adding the following line to the beginning of the inference script:

  ```
  import amct_tensorflow
  ```

---

4. (Optional) Convert the quantized model into an offline model adapted to the Ascend AI Processor by referring to [ATC Instructions](#).

### 5.3.1.2 Quantizing an Object Detection Network

**Prerequisites**

- **Model**
  
  Upload the TensorFlow model to be quantized to any directory on the Linux server as the AMCT installation user. This section uses the `yolov3/pre_model/yolov3_coco.pb` model available in the sample package as an example.

  **NOTE**

  If you choose to use your own model, you are advised to perform inference in the TensorFlow environment in advance to test if it can run properly in the TensorFlow environment with expected accuracy.

- **Image dataset**
  
  After the model is quantized using AMCT, perform inference with the model to test the model accuracy. Use the dataset that matches the model.

  Upload the dataset matching the model to any directory on the Linux server as the AMCT installation user. This section uses the `yolov3/detection.jpg` dataset available in the sample package as an example.

- **Calibration dataset**
  
  The calibration dataset is used to generate the quantization factors to guarantee the accuracy.

  The process of calculating the quantization factors is referred to as calibration. Perform inference using the quantized network with one or more batches of a subset of images from the validation dataset to complete calibration.
Upload the calibration dataset file to any directory on the Linux server as the AMCT installation user. This section uses the `yolov3/calibration.jpg` calibration dataset available in the sample package as an example.

**Procedure**

1. Go to the `sample/yolov3` directory and run the following command to quantize the YOLOv3 network:
   ```bash
   python3.7.5 yolov3_sample.py
   ```
   If messages similar to the following are displayed, the quantization is successful:
   ```plaintext
   INFO - [AMCT]:[save_model]: The model is saved in $HOME/amct/amct_tf/sample/yolov3/result/
   YOLOv3_quantized.pb     #Directory of the quantized model
   origin.png save successfully!      #Precheck result
   quantize.png save successfully!    #Postcheck result
   ```

2. After the quantization is complete, find the following files generated in the directory of the quantization script:
   - `config.json`: configuration file that describes how to quantize each layer in the model. If a quantization configuration file already exists in the directory of the quantization script, the existing quantization configuration file is overwritten by a new one with the same name in a call to 5.9.2.1 `create_quant_config`. If not that case, a new quantization configuration file is created.
   If the accuracy of model inference drops significantly after quantization, you can modify the `config.json` file by referring to 5.3.2 Quantization Configuration.
   - `record.txt`: file that records quantization factors. For details about the prototype definition of the file, see 5.11.4 Quantization Factor Record File.
   - `origin.png/quantize.png`: precheck and postcheck results.
   - `result`: quantization result folder, containing the quantized model file and the quantization information file.
     - `YOLOv3_quant.json`: quantization information file (named after the quantized model). This file gives the node mapping between the quantized model and the original model and is used for accuracy comparison between the quantized model and the original model.
     - `YOLOv3_quantized.pb`: quantized model that can serve for accuracy simulation in the TensorFlow environment and be deployed on the Ascend AI Processor.

   When a model is re-quantized, the existing result files will be overwritten.

3. (Optional) Convert the quantized deployable model into an offline model adapted to the Ascend AI Processor by referring to ATC Instructions.

### 5.3.1.3 Quantization Example Using the convert_model API

**Prerequisites**

- For details about how to prepare the model, image dataset, and calibration dataset, see 5.3.1.1.1 Quantization Preparations.
- Quantization factors:
  Upload the quantization factor record file to any directory on the Linux server as the AMCT installation user. The following uses the quantization factors of the MobileNetV2 network available in the sample package for illustration convenience. For details about quantization factors, see 5.11.4 Quantization Factor Record File.

Procedure

1. Run the quantization script to quantize the original network.

   The following uses the image classification network quantization script `MobileNetV2_convert_model.py`, the `mobilenet_v2.pb` model file, the `classification.jpg` dataset, and the `calibration` dataset available in the sample package to illustrate how to execute the quantization script.

   a. Run the following command in the `sample/mobilenetv2` directory to quantize the `mobilenet_v2.pb` network:

```
python3.7.5 MobileNetV2_convert_model.py
```

   If messages similar to the following are displayed, the quantization is successful:

```
INFO - [AMCT]:[save_model]: The model is saved in $HOME/amct/amct_tf/sample/mobilenetv2/ results/convert/MobileNetV2_quantized.pb     #Directory (including the file name) of the quantized model that serves for both deployment on the Ascend AI Processor and accuracy simulation in the TensorFlow environment.
```

   Origin Model Prediction:
```
category index: 699
category prob: 0.560       #Inference result of the original model. It is an example only.
```

   Quantized Model Prediction:
```
category index: 699
category prob: 0.569       #Inference result of the quantized model. It is an example only.
```

b. View the quantization result.

   Find the following files generated in the directory of the quantization script:

   - `results/convert`: quantization result directory, including:

     - `MobileNetV2_quant.json`: quantization information file. This file gives the node mapping between the quantized model and the original model and is used for accuracy comparison between the quantized model and the original model.
     - `MobileNetV2_quantized.pb`: quantized model that can both serve for accuracy simulation in the TensorFlow environment and be deployed on the Ascend AI Processor.

   When a model is re-quantized, the existing result files will be overwritten.

   c. (Optional) Convert the quantized model into an offline model adapted to the Ascend AI Processor by referring to:

```
ATC Instructions
```

5.3.2 Quantization Configuration

This section describes the quantization configuration file of image classification networks.
5.3.2.1 Overview

If the inference accuracy of the config.json quantization configuration file generated by the 5.9.2.1 create_quant_config call does not meet the requirements, you need to tune the config.json file until the accuracy is as expected. The following is an example of the file content.

- Uniform quantization configuration file

```json
{
  "version":1,
  "batch_num":1,
  "activation_offset":true,
  "joint_quant":false,
  "do_fusion":true,
  "skip_fusion_layers":[],
  "MobilenetV2/Conv/Conv2D":{
    "quant_enable":true,
    "activation_quant_params":{
      "max_percentile":0.999999,
      "min_percentile":0.999999,
      "search_range":{
        0.7,
        1.3
      },
      "search_step":0.01
    },
    "weight_quant_params":{
      "wts_algo":"arq_quantize",
      "channel_wise":true
    }
  },
  "MobilenetV2/Conv_1/Conv2D":{
    "quant_enable":true,
    "activation_quant_params":{
      "max_percentile":0.999999,
      "min_percentile":0.999999,
      "search_range":{
        0.7,
        1.3
      },
      "search_step":0.01
    },
    "weight_quant_params":{
      "wts_algo":"arq_quantize",
      "channel_wise":true
    }
  },
  "MobilenetV2/Logits/AvgPool":{
    "quant_enable":true,
    "activation_quant_params":{
      "max_percentile":0.999999,
      "min_percentile":0.999999,
      "search_range":{
        0.7,
        1.3
      },
      "search_step":0.01
    },
    "weight_quant_params":{
      "wts_algo":"arq_quantize",
      "channel_wise":false
    }
  }
}
```
### 5.3.2.2 Configuration File Options

The following tables describe the parameters in the configuration file.

#### Table 5-8 version

<table>
<thead>
<tr>
<th>Function</th>
<th>Version number of the quantization configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>Currently, only version 1 is available.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>1</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

#### Table 5-9 batch_num

<table>
<thead>
<tr>
<th>Function</th>
<th>Batch count for quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>Greater than 0</td>
</tr>
<tr>
<td>Description</td>
<td>If this option is not set, the default value 1 is used. It is recommended that the number of images in the calibration dataset be less than or equal to 50. The value of batch_num is calculated based on the value of batch_size. batch_num x batch_size equals the number of images in the calibration dataset used for quantization. batch_size indicates the number of images per batch.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>1</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

#### Table 5-10 activation_offset

<table>
<thead>
<tr>
<th>Function</th>
<th>Symmetric quantization or asymmetric quantization select for activation quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
</tbody>
</table>
If it is set to **true**, asymmetric quantization is used. If it is set to **false**, symmetric quantization is used.

<table>
<thead>
<tr>
<th>Description</th>
<th>Eltwise joint quantization switch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td><strong>true</strong> or <strong>false</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If it is set to <strong>true</strong>, Eltwise joint quantization is enabled. If it is set to <strong>false</strong>, Eltwise joint quantization function is disabled.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>false</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

If it is set to **true**, BN fusion is enabled. If it is set to **false**, BN fusion is disabled.

<table>
<thead>
<tr>
<th>Description</th>
<th>Layer skip in BN fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>string</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>Layers to skip in BN fusion. For details about the supported fusion patterns, see 5.1.2.5 Fusion Support.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>BN fusion switch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td><strong>true</strong> or <strong>false</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If it is set to <strong>true</strong>, BN fusion is enabled. If it is set to <strong>false</strong>, BN fusion is disabled.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
<tr>
<td>Description</td>
<td>Layers to skip in BN fusion</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 5-14 layer_config**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization configuration of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>object</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Includes the following parameters:</td>
</tr>
<tr>
<td></td>
<td>● quant_enable</td>
</tr>
<tr>
<td></td>
<td>● activation_quant_params</td>
</tr>
<tr>
<td></td>
<td>● weight_quant_params</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 5-15 quant_enable**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization enable by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>true or false</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If it is set to true, the layer is to be quantized. If it is set to false, otherwise.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 5-16 activation_quant_params**

<table>
<thead>
<tr>
<th>Function</th>
<th>Activation quantization parameters of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>object</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>None</td>
</tr>
<tr>
<td>-----------------</td>
<td>------</td>
</tr>
</tbody>
</table>
| **Description** | Includes the following parameters:  
- max_percentile  
- min_percentile  
- search_range  
- search_step |
| **Recommended Value** | None |
| **Required/Optional** | Optional |

**Table 5-17 weight_quant_params**

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th>Weight quantization parameters of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>object</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
| **Description** | Includes the following parameters in uniform quantization:  
- wts_algo  
- channel_wise |
| **Recommended Value** | None |
| **Required/Optional** | Optional |

**Table 5-18 max_percentile**

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th>Upper search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>float</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>(0.5, 1]</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Indicates the maximum number to be considered as the search result among a group of numbers in descending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the maximum, that is, the largest number. A larger value indicates that the upper clip limit is closer to the maximum value of the data to be quantized.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>0.999999</td>
</tr>
</tbody>
</table>
### Table 5-19 min_percentile

<table>
<thead>
<tr>
<th>Function</th>
<th>Lower search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0.5, 1]</td>
</tr>
<tr>
<td>Description</td>
<td>Indicates the minimum number to be considered as the search result among a group of numbers in ascending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the minimum, that is, the smallest number. A larger value indicates that the lower clip limit is closer to the minimum value of the data to be quantized.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>0.999999</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 5-20 search_range

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search range: [search_range_start, search_range_end]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>A list of two floats</td>
</tr>
<tr>
<td>Value Range</td>
<td>0 &lt; search_range_start &lt; search_range_end</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the quantization factor search range:</td>
</tr>
<tr>
<td></td>
<td>- search_range_start: search start.</td>
</tr>
<tr>
<td></td>
<td>- search_range_end: search end.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>[0.7, 1.3]</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 5-21 search_step

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0, (search_range_end – search_range_start)]</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the quantization factor search step. A smaller value indicates a smaller step.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>0.01</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 5-22 wts_algo**

<table>
<thead>
<tr>
<th>Function</th>
<th>Weight quantization algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value Range</td>
<td>arq_quantize</td>
</tr>
<tr>
<td>Description</td>
<td>arq_quantize: uniform quantization</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>None</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 5-23 channel_wise**

<table>
<thead>
<tr>
<th>Function</th>
<th>Whether to use different quantization factors for each channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
</tbody>
</table>
| Description  | • If it is set to true, channels are separately quantized using different quantization factors.  
                          • If it is set to false, all channels are quantized altogether using the same quantization factors. |
| Recommended Value | true                                           |
| Required/Optional | Optional                                |

### 5.3.2.3 Configuration Tuning

If the inference accuracy of the model quantized based on the default `config.json` configuration drops significantly, perform the following steps to tune the quantization configuration file. The following provides instructions on manually
tuning the configuration file. To use the automatic tuning feature provided by AMCT, see 5.5 Accuracy-oriented Automatic Quantization Rollback.

Step 1 Execute the quantization script in the amct_tensorflow_sample.tar.gz package to perform quantization based on the default configuration generated by the 5.9.2.1 create_quant_config API.

Step 2 If the inference accuracy with the model quantized in Step 1 is as expected, configuration tuning ends. Otherwise, go to Step 3.

Step 3 Tune batch_num in the quantization configuration file.

batch_num controls the batch count for quantization. Tune it based on the batch size and the number of images required for quantization. Generally, a larger quantity of data samples used in a quantization process indicates a smaller accuracy loss after quantization. However, excessive data does not necessarily improve accuracy, but certainly consumes more memory and reduces the quantization speed, hence resulting in insufficient memory, video RAM, and thread resources. Therefore, it is recommended that the product of batch_num and batch_size be 16 or 32.

Step 4 If the inference accuracy with the model quantized in Step 3 is as expected, configuration tuning ends. Otherwise, go to Step 5.

Step 5 Tune quant_enable in the quantization configuration file.

quant_enable specifies whether to quantize a layer. If it is set to true, the layer is to be quantized. If it is set to false, otherwise. If the configuration of a layer is not present, the quantization of the layer is skipped. Generally, specifying fewer layers to quantize improves quantization accuracy. When the network accuracy is not as expected, locate the quantization-sensitive layers (whose error increases significantly after quantization, such as the top layer, bottom layer, depthwise convolutional layer, and layers with few parameters) in the network, and disable quantization on these layers as needed.

Step 6 If the inference accuracy with the model quantized in Step 5 is as expected, configuration tuning ends. Otherwise, go to Step 7.

Step 7 Tune the values of activation_quant_params and weight_quant_params in the quantization configuration file.

- Data is clipped to the range [left, right] specified by the activation_quant_params parameters. Generally, values distributed near a boundary are sparse, and clip may be performed on all the values, to improve the accuracy. A larger value of min_percentile (max_percentile) indicates that left (right) is closer to the minimum value (maximum value) of the to-be-quantized data. search_range and search_step affect the range of [left, right]. Generally, a larger value of search_range and a smaller value of search_step may achieve higher quantization accuracy, but the quantization takes more time.

- channel_wise in weight_quant_params determines whether to use a different quantization factor for each channel during weight quantization. If it is set to true, channels are separately quantized using different quantization factors. If it is set to false, all channels are quantized altogether using the same quantization factors. Generally, the inference accuracy is higher if the channels are quantized separately. However, the MatMul and AvgPool layers
are channel-irrelevant. Therefore, this parameter does not take effect on these layers.

**Step 8** If the inference accuracy with the model quantized in **Step 7** is as expected, configuration tuning ends. Otherwise, it indicates that quantization has severe adverse impact on the inference accuracy. In this case, remove the quantization configuration.

----End
5.4 Quantization Aware Training
5.4.1 Quantization Example

Prerequisites

- Model
  Upload a pre-trained TensorFlow model to any directory on the Linux server as the AMCT installation user.
  The following uses the sample/resnet_v1_50 model as an example. The pre_model/ResNet50_train.meta file is used for quantization aware training. The pre_model/ResNet50_eval.meta file is used for validation.

- Image dataset
  Because quantization aware training needs huge data to further optimize the quantization parameters, keep the data consistent with that used to train the model. The ResNet-50 dataset is trained on the ILSVRC-2012-CLS subset of ImageNet. Therefore, you need to prepare a dataset in TFRecord format. To use a different dataset, you need to preprocess the data by yourself.

Procedure

1. Perform quantative aware training on the resnet_v1_50 model.
   Run the following command in the sample/resnet_v1_50 directory:
   ```bash
   python3.7.5 resnet_v1_50_retrain_sample.py --config_defination CONFIG_DEFINATION --batch_num BATCH_NUM --train_set TRAIN_SET [--train_keyword TRAIN_KEYWORD] --eval_set EVAL_SET [--eval_keyword EVAL_KEYWORD] --train_model TRAIN_MODEL --eval_model EVAL_MODEL
   ```
   The following table describes the available command-line options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>• Optional.</td>
</tr>
<tr>
<td></td>
<td>• Displays help information.</td>
</tr>
<tr>
<td>--config_defination CONFIG_DEFINATION</td>
<td>• Optional.</td>
</tr>
<tr>
<td></td>
<td>• A string.</td>
</tr>
<tr>
<td></td>
<td>• Default: None</td>
</tr>
<tr>
<td></td>
<td>• Sets the directory of the simplified</td>
</tr>
<tr>
<td></td>
<td>quantization configuration file.</td>
</tr>
<tr>
<td>--batch_num BATCH_NUM</td>
<td>• Optional.</td>
</tr>
<tr>
<td></td>
<td>• An int.</td>
</tr>
<tr>
<td></td>
<td>• Default: 2</td>
</tr>
<tr>
<td></td>
<td>• Sets the batch count in the</td>
</tr>
<tr>
<td></td>
<td>inference phase of quantization</td>
</tr>
<tr>
<td></td>
<td>aware training.</td>
</tr>
</tbody>
</table>

Table 5-24 Quantization aware training command-line options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| --train_set TRAIN_SET  | ● Required.  
● A string.  
● Default: None  
● Sets the directory of the image dataset.                                                                                                           |
| --train_keyword TRAIN_KEYWORD | ● Optional.  
● A string.  
● Default: None  
● Selects files containing the keyword in the training dataset directory. If it is not specified, all files in the training dataset directory are used by default. |
| --eval_set EVAL_SET    | ● Required.  
● A string.  
● Default: None  
● Sets the directory of the validation dataset.                                                                                                                                                      |
| --eval_keyword EVAL_KEYWORD | ● Optional.  
● A string.  
● Default: None  
● Selects files containing the keyword in the validation dataset directory. If it is not specified, all files in the validation dataset directory are used by default.                          |
| --train_model TRAIN_MODEL | ● Required.  
● A string.  
● Default: ./pre_model/resnet_v1_50_train.meta  
● Sets the model directory.                                                                                                                  |
| --eval_model EVAL_MODEL | ● Required.  
● A string.  
● Default: ./pre_model/resnet_v1_50_eval.meta  
● Sets the directory of the model for validation.                                                                                              |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| --num_parallel_reads NUM_PARALLEL_READS | • Optional.  
• An int.  
• Default: 4  
• Sets the number of threads for reading datasets. Set the argument according to the hardware compute capability. |
| --buffer_size BUFFER_SIZE      | • Optional.  
• An int.  
• Default: 1000  
• Sets the size for dataset out-of-order buffer. Set the argument to the memory capacity. |
| --repeat_count REPEAT_COUNT    | • Optional.  
• An int.  
• Default: 0  
• Sets the number of repetitions of the dataset. The argument 0 indicates infinite repetition. |
| --batch_size BATCH_SIZE        | • Optional.  
• An int.  
• Default: 32  
• Sets the batch size of TensorFlow execution. Set the argument according to the memory or video RAM capacity. |
| --ckpt CKPT_PATH               | • Optional.  
• A string.  
• Default: None  
• Sets the directory of the benchmark weight checkpoint file of the ResNet-50 v1 model. If it is not specified, the file is automatically downloaded. |
| --learning_rate LEARNING_RATE  | • Optional.  
• A float.  
• Default: 1e-6  
• Sets the learning rate. |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| --save_interval SAVE_INTERVAL | • Optional.  
  • An int.  
  • Default: 1000  
  • Sets the save interval. |
| --momentum MOMENTUM   | • Optional.  
  • A float.  
  • Default: 0.9  
  • Sets the momentum of the RMSPropOptimizer. |
| --train_iter TRAIN_ITER | • Optional.  
  • An int.  
  • Default: 1000  
  • Sets the number of training iterations. |

An example is as follows:

```python
python3.7.5 resnet_v1_50_retrain_sample.py --config_definition sample.cfg --train_set /data/Datasets/imagenet_TF_trainset/ --eval_set /data/Datasets/imagenet_TF_evalset/
```

**NOTICE**

If the upload directory of the test dataset is the same as that of the validation dataset, add the `--train_keyword TRAIN_KEYWORD` and `--eval_keyword EVAL_KEYWORD` options to the quantization command to ensure that the `--train_set` option points to the test dataset and the `--eval_set` option points to the validation dataset.

If messages similar to the following are displayed, the quantization aware training is successful:

The model after retraining top 1 accuracy = 68.26%.  
The model after retraining top 5 accuracy = 87.954%.

2. View the retrain result.

After the quantization aware training is complete, find the following folders and files generated to the `sample/resnet_v1_50` directory:

- `retrain/results/resnet_v1_50_quantized.pb`: result model file of quantization aware training. This model can be used for accuracy simulation in the TensorFlow environment and can also be deployed on the Ascend AI Processor.
- `tmp`: quantization aware training temporary directory, containing:
  - `checkpoint`: checkpoints saved during TensorFlow training.
- **config.json**: quantization aware training configuration file that describes how to train each layer in the model. If a quantization aware configuration file already exists in the directory of the quantization aware script, the existing configuration file is overwritten by a new one with the same name in a call to **5.9.3.1 create_quant_retrain_config**. If not that case, a new configuration file is created.

- **events.out.tfevents.xxxxxxxxxx.xxx**: file that describes the quantization aware training loss information, which can be viewed on the TensorBoard.

- **record.txt**: file that records quantization factors. For details about the prototype definition of the file, see **5.11.4 Quantization Factor Record File**.

- **resnet_v1_50.pb**: result model file.

- **resnet_v1_50_retrain-0.data-00000-of-00001**: weight file generated after quantization aware training.

- **resnet_v1_50_retrain-0.index**: index of the weight file generated after quantization aware training.

- **resnet_v1_50_retrain-0.meta**: model file to be retrained in the TensorFlow environment.

- **resnet_v1_50_retrain-1.data-00000-of-00001**: weight file generated after quantization aware training.

- **resnet_v1_50_retrain-1.index**: index of the weight file generated after quantization aware training.

- **resnet_v1_50_retrain-1.meta**: model file to be retrained in the TensorFlow environment.

When quantization aware training is run again on the model, the existing result files will be overwritten.

### 5.4.2 Quantization Configuration

#### 5.4.2.1 Overview

If inference based on the **config.json** quantization aware training configuration file generated by the **5.9.3.1 create_quant_retrain_config** call has significant accuracy drop, tune the **config.json** file until the accuracy is as expected. The following is an example of the file content. Keep the layer names unique in the JSON file.

```json
{
  "version":1,
  "batch_num":1,
  "conv7":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo":"ulq_quantize"
    }
  }
}
```
5.4.2.2 Configuration File Options

The following describes the configuration options available in the configuration file.

**Table 5-25** version

<table>
<thead>
<tr>
<th>Function</th>
<th>Version number of the quantization configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>Currently, only version 1 is available.</td>
</tr>
<tr>
<td>Recommended</td>
<td>1</td>
</tr>
<tr>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 5-26** batch_num

<table>
<thead>
<tr>
<th>Function</th>
<th>Batch count in the inference phase of quantization aware training.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>Greater than 0</td>
</tr>
</tbody>
</table>
If this option is not set, the default value 1 is used. It is recommended that the number of images in the calibration dataset be less than or equal to 50. The value of `batch_num` is calculated based on the value of `batch_size`. 

`batch_num` x `batch_size` equals the number of images in the calibration dataset used for quantization. 

`batch_size` indicates the number of images per batch.

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>If set to true, quantization aware training is performed at this layer. If set to false, otherwise.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 5-27 retrain_enable

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization aware training enable by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td>If set to true, quantization aware training is performed at this layer. If set to false, otherwise.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 5-28 retrain_data_config

<table>
<thead>
<tr>
<th>Function</th>
<th>Activation quantization configuration by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
</tbody>
</table>
| Description | Includes the following parameters:  
- **algo**: selects the quantization algorithm, default to `ulq_quantize`.  
- **clip_max**: sets the upper limit of the clip quantization algorithm, default to be empty.  
- **clip_min**: sets the lower limit of the clip quantization algorithm, default to be empty.  
- **fixed_min**: fixes the minimum value of the clip quantization algorithm at 0, default to be empty. |

<table>
<thead>
<tr>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
</tr>
<tr>
<td>Required/Optional</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
</tbody>
</table>

**Table 5-29 retrain_weight_config**

<table>
<thead>
<tr>
<th>Function</th>
<th>Weight quantization configuration by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>Includes the following parameters:</td>
</tr>
<tr>
<td></td>
<td>● <strong>algo</strong>: quantization algorithm select, default to <strong>arq_retrain</strong>.</td>
</tr>
<tr>
<td></td>
<td>● <strong>channel_wise</strong></td>
</tr>
<tr>
<td>Recommended Value</td>
<td>None</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 5-30 algo**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization algorithm by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>● <strong>ulq_quantize</strong>: ULQ clip quantization algorithm.</td>
</tr>
<tr>
<td></td>
<td>● <strong>arq_retrain</strong>: ARQ quantization algorithm.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>Set to <strong>ulq_quantize</strong> for activation quantization or <strong>arq_retrain</strong> for weight quantization.</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 5-31 channel_wise**

<table>
<thead>
<tr>
<th>Function</th>
<th>Whether to use different quantization factors for each channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td><strong>true</strong> or <strong>false</strong></td>
</tr>
<tr>
<td>Description</td>
<td>● If set to <strong>true</strong>, channels are separately quantized using different quantization factors.</td>
</tr>
<tr>
<td></td>
<td>● If set to <strong>false</strong>, all channels are quantized altogether using the same quantization factors.</td>
</tr>
<tr>
<td>Table 5-32 fixed_min</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>--</td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td>Lower limit enable of the activation quantization algorithm</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td><strong>true</strong> or <strong>false</strong></td>
</tr>
</tbody>
</table>
| **Description**      | ● If set to **true**, the lower limit of the activation quantization algorithm is fixed at 0.  
                           ● If set to **false**, the lower limit of the activation quantization algorithm is not fixed.  
                           If this option is not included, AMCT automatically sets the lower limit of the activation quantization algorithm according to the graph structure.  
                           If this option is included: when the upstream layer of the quantization layer is ReLU, you need to manually set this option to **true**; when the upstream layer of the quantization layer is not ReLU, you need to manually set this option to **false**. |
| **Recommended Value**| Do not include this option. |
| **Required/Optional**| Optional |

<table>
<thead>
<tr>
<th>Table 5-33 clip_max</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>Upper limit of the activation quantization algorithm</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>float</td>
</tr>
</tbody>
</table>
| **Value Range**     | clip_max>0  
                          Controls the upper limit max based on the data distribution of the activations at different layers. The recommended value range is as follows:  
                          [0.3 * max, 1.7 * max] |
| **Description**     | If this option is included, the clip upper limit of the activation quantization algorithm is fixed. If this option is not included, the clip upper limit is learned using the IFMR algorithm. |
| **Recommended Value**| Do not include this option. |
| **Required/Optional**| Optional |
Table 5-34 clip_min

<table>
<thead>
<tr>
<th>Function</th>
<th>Lower limit of the activation quantization algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>clip_min &lt; 0</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the lower limit min based on the data</td>
</tr>
<tr>
<td></td>
<td>distribution of the activations at different layers.</td>
</tr>
<tr>
<td></td>
<td>The recommended value range is as follows:</td>
</tr>
<tr>
<td></td>
<td>[0.3 * min, 1.7 * min]</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>Do not include this option.</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

5.4.2.3 Configuration Tuning

If the inference accuracy of the model quantized based on the default config.json configuration drops significantly, perform the following steps to tune the quantization configuration file. The following provides instructions on manually tuning the configuration file. To use the automatic tuning feature provided by AMCT, see 5.5 Accuracy-oriented Automatic Quantization Rollback.

1. Execute the quantization script in the amct_tensorflow_sample.tar.gz package to perform quantization based on the default configuration generated by the 5.9.3.1 create_quant_retrain_config API. If the quantization accuracy is as expected, the configuration tuning ends. Otherwise, go to 2.

2. Cancel the quantization of certain layers by changing the value of retrain_enable to false. Generally, the input and output layers of a model have a greater impact on the inference result. Therefore, you can try to cancel the quantization of the input and output layers first.

If you have desirable settings for clip_max and clip_min, modify the quantization configuration file as follows.

```json
{
    "version":1,
    "batch_num":1,
    "inference/Conv2D":{
        "retrain_enable":true,
        "retrain_data_config":{
            "algo":"ulq_quantize",
            "clip_max":3.0,
            "clip_min":-3.0
        },
        "retrain_weight_config":{
            "algo":"arq_retrain",
            "channel_wise":true
        }
    },
    "inference/Conv2D_1":{
        "retrain_enable":true,
        "retrain_data_config":{
            "algo":"arq_retrain",
            "channel_wise":true
        }
    }
}
```
3. Configuration tuning ends if the inference accuracy meets the requirement. Otherwise, it indicates that quantization aware training has severe adverse impact on the inference accuracy. In this case, remove the quantization aware training configuration.

### 5.5 Accuracy-oriented Automatic Quantization Rollback

#### 5.5.1 Prerequisites

**Quantization Code Modification**

Manually implement a class inherited from `AutoCalibrationEvaluatorBase` and rewrite the three methods under the base class.

1. Implement the `evaluate` method inherited from the `model_evaluator` base class. Keep the inputs and return of the function consistent with those of the base class. The function should return a unique model accuracy metric, for example, top 1 accuracy results for image classification networks.

2. Implement the `calibration` method inherited from the `model_evaluator` base class. Keep the inputs and return of the function consistent with those of the base class. This method is used to run model inference during the calibration process. Keep the number of inference batches the same as the `batch_num` argument.

3. Implement the `metric_eval` method inherited from the `model_evaluator` base class. Keep the inputs and return of the function consistent with those of the base class. This method is used to test the accuracy of the quantized model during automatic quantization configuration search. It returns a tuple, where the first element indicates whether the accuracy is acceptable and the second element indicates the accuracy drop.

The sample code is provided as follows. Find the code file in/amct_tensorflow/common/auto_calibration/auto_calibration_evaluator_base.py in the AMCT installation path.

```python
class AutoCalibrationEvaluatorBase:
    """the base class for ModelEvaluator""
    def __init__(self):
        """Function:__init__ function of class"
        pass

def calibration(self, model): # pylint: disable=R0201
Model

Prepare a TensorFlow model (.pb format). The following uses the `mobilenet_v2.pb` model available in the sample package as an example.

Dataset

Prepare a validation dataset for calibrating and testing the model during automatic quantization rollback. The ILSVRC-2012-CLS subset of ImageNet is used in this example, which is in TFRecord format and contains 50,000 images in total.

If you choose to prepare your own dataset, you need to modify the data preprocessing part in the sample code to match the model inputs.

5.5.2 Rollback Example

1. Run the quantization script.
Go to the `sample/mobilenetv2` directory and run the following command to perform accuracy-oriented automatic quantization rollback. The accuracy drop limit is 0.5% by default.

```
python3.7.5 --dataset DATASET
```

The command-line options are described as follows.

**Table 5-35 Command-line options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| -h                         | ● Optional.  
|                            | ● Displays help information.                                                 |
| --dataset DATASET          | ● Required.  
|                            | ● A string.  
|                            | ● Default: **None**  
|                            | ● Specifies the validation dataset (the ILSVRC-2012-CLS subset of ImageNet in TFRecord format). |
| --num_parallel_reads NUM_PARALLEL_READS | ● Optional.  
|                            | ● An int.  
|                            | ● Default: **4**  
|                            | ● Sets the number of threads for reading datasets. Set the argument according to the hardware compute capability. |
| --batch_size BATCH_SIZE    | ● Optional.  
|                            | ● An int.  
|                            | ● Default: **32**  
|                            | ● Sets the batch size of TensorFlow execution. Set the argument according to the memory or video RAM capacity. |
| --model MODEL              | ● Optional.  
|                            | ● A string.  
|                            | ● Default: `pre_model/mobilenet_v2.pb`  
|                            | ● Sets the original model for automatic quantization rollback. |

An example is as follows:

```
python3.7.5 --dataset /data/Datasets/imagenet_TF_evalset/
```

If messages similar to the following are displayed, the model is successfully quantized. (The inference accuracy results are examples only.)

```
# Accuracy of the original model
2021-01-09 18:25:36,689 - INFO - [AMCT]:[AMCT]: Accuracy of original model is 74.97
```

Issue 01 (2021-07-01)  Copyright © Huawei Technologies Co., Ltd.  229
2. View the result.

After successful automatic quantization rollback, the accuracy of the original model, the accuracy of the fake-quantized model based on the default quantization configuration, and the accuracy of the fake-quantized model generated after automatic quantization rollback are displayed. Find the following files generated in the sample/mobilenetv2 directory:

- **amct_log**:

- **results/accuracy_based_auto_calibration**:
  - accuracy_based_auto_calibration_final_config.json: quantization configuration file generated after accuracy-oriented automatic quantization rollback.
  - accuracy_based_auto_calibration_ranking_information.json: file that records the quantization sensitivity information of layers to be quantized.
  - config.json: original quantization configuration file.
  - MobileNetV2_quantized.pb: quantized model that can both serve for accuracy simulation in the TensorFlow environment and be deployed on the Ascend AI Processor, generated after accuracy-oriented automatic quantization rollback.
  - MobileNetV2_quant.json: file that records graph fusion operations.
  - record.txt: file that records quantization factors generated during quantization. For details about the prototype definition of the file, see 5.11.4 Quantization Factor Record File.

When you run automatic quantization rollback again, the existing result files will be overwritten.

### 5.6 Tensor Decomposition
5.6.1 Decomposition Prerequisites

Restrictions

If a Conv2D layer uses a large shape, the decomposition will be time-consuming or terminated abnormally. To prevent this problem, refer to the following before starting decomposition:

- **Reference performance specifications of the decomposition tool:**
  - CPU: Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20 GHz
  - At least 512 GB memory
  Time taken to decompose a single Convolution layer:
  - About 25s for shape (512, 512, 5, 5).
  - About 16s for shape (1024, 1024, 3, 3).
  - About 78s for shape (1024, 1024, 5, 5).
  - About 63s for shape (2048, 2048, 3, 3).
  - About 430s for shape (2048, 2048, 5, 5).

- **Memory consideration:**
  It takes about 32 GB memory to decompose the Convolution kernel with shape (2048, 2048, 5, 5).

Prerequisites

Upload the TensorFlow model file to be decomposed and its weight files to any directory on the Linux server. The model file is a .meta file, and the weight file consists of a .data-0000X-of-0000X file and an .index weight file index. (The .data-0000X-of-0000X weight file and the .index weight file index are used in pair.)

This follows uses the training script to generate a model file and the weight files (.data-0000X-of-0000X and .index).

**Step 1** Prepare a training dataset.

- If the server installed with AMCT has Internet access:
  Create a dataset directory in the sample/tensor_decompose/ directory on the server, for example, data/mnist.
  When the script is executed in **Step 2**, the training and test datasets are automatically downloaded to the data/mnist directory from [http://yann.lecun.com/exdb/mnist/](http://yann.lecun.com/exdb/mnist/).

- If the user environment has no Internet access:
  On a server with Internet access, visit the following links to download the corresponding packages and upload them to the sample/tensor_decompose/data/mnist directory on the AMCT server. (If the data/mnist directory does not exist, create it first.)
  b. The label file of the MNIST training dataset is available at: [http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz)
c. The MNIST test dataset is available at: http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz

d. The label file of the MNIST test dataset is available at: http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz

Step 2 Execute the training script to generate the model file and weight files.

The sample/tensor_decompose directory provides two copies of TensorFlow training code, one using the Session API and the other using the Estimator API. You can select one that best suits your needs. Go to the sample/tensor_decompose/ directory and run the following command to generate the model file and weight files:

```python
python3.7.5 train_sample_session.py --data_path DATA_PATH
```

Alternatively,

```python
python3.7.5 train_sample_estimator.py --data_path DATA_PATH
```

`--data_path` is a required option that specifies the directory of the MNIST dataset, which can be absolute or relative.

An example is as follows:

```python
python3.7.5 train_sample_session.py --data_path=data/mnist
```

If messages similar to the following are displayed, the execution is successful:

Valid Accuracy: 0.9803 // Training accuracy based on the MNIST dataset

Step 3 After the execution is successful, the model file and its weight files are generated in the sample/tensor_decompose/checkpoints directory. The following is an example:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Size</th>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkpoint</td>
<td>85 B</td>
<td>04:37</td>
<td>85 Jul 28 04:37 checkpoint</td>
</tr>
<tr>
<td>model.ckpt-200.index</td>
<td>771 B</td>
<td>04:37</td>
<td>771 Jul 28 04:37 model.ckpt-200.index</td>
</tr>
<tr>
<td>model.ckpt-200.meta</td>
<td>60937 B</td>
<td>04:37</td>
<td>60937 Jul 28 04:37 model.ckpt-200.meta</td>
</tr>
</tbody>
</table>

-----End

5.6.2 Decomposition Example

Procedure

Step 1 Go to the amct_tf/sample/tensor_decompose directory and execute the script for decomposing the original model:

```python
python3.7.5 decompose_sample.py --meta_path META_PATH --ckpt_path CKPT_PATH --save_path SAVE_PATH
```

Table 5-36 describes the command-line options.
Table 5-36 Command-line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--meta_path</td>
<td>(Required) Directory of the TensorFlow model file (.meta).</td>
</tr>
<tr>
<td>CKPT_PATH</td>
<td>(Required) Directory of the TensorFlow model weight files, including .data-0000X-of-0000X and .index files. The directory does not contain the file name extensions.</td>
</tr>
<tr>
<td>--save_path</td>
<td>(Required) Directory of the result files after tensor decomposition, which can be relative or absolute. If the specified directory does not exist, the directory is automatically created. The directory must be prefixed with the name of the result model, for example, tmp/model_decomposition.</td>
</tr>
</tbody>
</table>

An example is as follows:

```python3.7.5 decompose_sample.py --meta_path checkpoints/model.ckpt-200.meta --ckpt_path checkpoints/model.ckpt-200 --save_path tmp/model_decomposition
```

During the decomposition, the names of the decomposition-capable operators and the names of the decomposed operators are recorded in the log. The following log messages are examples only.

```
[AMCT]:[AMCT]: Processing conv2d_1/Conv2D
[AMCT]:[AMCT]: Decompose conv2d_1/Conv2D -> ['conv2d_1/Conv2D/decom_first/decom_first', 'conv2d_1/Conv2D/decom_core/decom_core', 'conv2d_1/Conv2D/decom_last/decom_last']
...
```

If messages similar to the following are displayed, the decomposition is successful:

```
auto_decomposition complete!
```

**Step 2** After the decomposition is complete, the result model is generated in the path specified by the --save_path option.

```
-rw-r--r-- 1 amct amct 95 Jul 28 04:49 checkpoint                       //Checkpoint when the result model is generated
-rw-r--r-- 1 amct amct 10846856 Jul 28 04:49 model_decomposition.data-00000-of-00001 //Weight file of the result model
-rw-r--r-- 1 amct amct 967 Jul 28 04:49 model_decomposition.index        //Weight file index of the result model
-rw-r--r-- 1 amct amct 315800 Jul 28 04:49 model_decomposition.meta       //Result model file
-rw-r--r-- 1 amct amct 517 Jul 28 04:49 model_decomposition.pkl           //Modifications made to the graph structure, used by the decompose_graph API
```

-----End

**Fine-tuning**

In normal cases, the accuracy of a decomposed model is lower than that of the original model. Therefore, fine-tuning is introduced to improve the accuracy of the decomposed model. Decrease the learning rate from about 0.1 times of the original learning rate. The number of epochs varies with models. The more convolutional layers are decomposed, the more epochs are required. The fine-
tuned model can have accuracy improvement. However, it is also possible that the accuracy remains or even drops.

- **Prerequisites**
  In this section, the 5.9.4.2 decompose_graph API is called and then inserted into the training code to fine-tune the decomposed model. Ensure that the model and weight returned by the 5.9.4.1 auto_decomposition call are based on the exact training code. The sample/tensor_decompose directory provides two copies of TensorFlow training code, one using the Session API and the other using the Estimator API. You can select one that best suits your needs.

- **Example**
  Insert the training code by calling the 5.9.4.2 decompose_graph API after the model is built and before the optimizer is built. For details, see finetune_sample_session.py and finetune_sample_estimator.py. After this API call, load the weight of the decomposed model for fine-tuning. The CKPT file generated after decomposition may contain the optimizer parameters of the model before decomposition, which do not match the decomposed model. Therefore, load only the weight of the decomposed model, rather than the optimizer parameters. For details, see finetune_sample_session.py and finetune_sample_estimator.py.

  a. Go to the amct_tf/sample/tensor_decompose directory and run the following command to fine-tune the model:

  ```
  python3.7.5 finetune_sample_session.py --data_path DATA_PATH --save_path SAVE_PATH
  ```

  Alternatively,

  ```
  python3.7.5 finetune_sample_estimator.py --data_path DATA_PATH --save_path SAVE_PATH
  ```

  Table 5-37 describes the command-line options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--data_path DATA_PATH</td>
<td>(Required) Directory of the MNIST dataset. See Prerequisites.</td>
</tr>
<tr>
<td>--save_path SAVE_PATH</td>
<td>(Required) Directory of the result files generated by the 5.9.4.1 auto_decomposition call.</td>
</tr>
</tbody>
</table>

An example is as follows:

```
python3.7.5 finetune_sample_session.py --data_path data/mnist --save_path tmp/model_decomposition
```

**NOTE**

- If the script used for generating the model file and weight files in Prerequisites is train_sample_session.py, use the finetune_sample_session.py fine-tuning script.
- If the script used for generating the model file and weight files in Prerequisites is train_sample_estimator.py, use the finetune_sample_estimator.py fine-tuning script.

If messages similar to the following are displayed, the execution is successful:

```
b. After fine-tuning is complete, the finetuned_ckpt result file is automatically generated in the sample/tensor_decompose directory that stores the model file and weight files of the decomposed model. The following is an example.

- rw-r--r-- 1 amct amct 128 Aug 1 04:14 checkpoint
  Index of the weight file after -rw-r--r-- 1 amct amct 10093192 Aug 1 04:14 model.ckpt-100.data-00000-of-00001  //Weight file index of the fine-tuned model
- rw-r--r-- 1 amct amct 1144 Aug 1 04:14 model.ckpt-100.index  //Weight file of the fine-tuned model
- rw-r--r-- 1 amct amct 122783 Aug 1 04:14 model.ckpt-100.meta  //Fine-tuned model
- rw-r--r-- 1 amct amct 3171658 Aug 1 04:14 model.pb  //.pb file of the fine-tuned model

For details about the files generated after networks from more open-source frameworks are decomposed, see 5.11.5 Tensor Decomposition Specification Reference of Open-Source Networks.

c. For details about how to quantize a model in PB format, see 5.3 Post-training Quantization or 5.4 Quantization Aware Training.

5.7 AMCT Update

The latest AMCT release allows you to access the latest features. Before updating AMCT, uninstall the existing installation by referring to 5.8 AMCT Uninstallation, and then install the latest version by referring to 5.2 AMCT Installation.

5.8 AMCT Uninstallation

You can uninstall AMCT as follows:

1. Run the following command in any directory on the Linux server as the AMCT installation user:
   pip3.7.5 uninstall amct_tensorflow

2. When the following information is displayed, enter y:
   Uninstalling amct-tensorflow-<version>.
   Would remove:
   ...
   ...
   Proceed (y/n)? y

   If a message similar to the following is displayed, the uninstallation is successful.
   Successfully uninstalled amct-tensorflow-<version>

   The installed TensorFlow will not be uninstalled during the uninstallation.

5.9 API Description

5.9.1 Common APIs
5.9.1.1 set_logging_level

Description
Sets the logging levels of the log messages printed to the screen and those saved to the `amct_log/amct_tensorflow.log` file.

Prototype

```
set_logging_level(print_level='info', save_level='info')
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_level</td>
<td>Input</td>
<td>Logging level of the log messages printed to the screen.</td>
<td>Default: info&lt;br&gt;A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>debug</strong>: DEBUG, INFO, WARNING, and ERROR logs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>info</strong>: INFO, WARNING, and ERROR logs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>warning</strong>: WARNING and ERROR logs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>error</strong>: ERROR logs.</td>
<td></td>
</tr>
<tr>
<td>save_level</td>
<td>Input</td>
<td>Logging level of log messages saved to the <code>quant_info.log</code> file.</td>
<td>Default: info&lt;br&gt;A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>debug</strong>: DEBUG, INFO, WARNING, and ERROR logs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>info</strong>: INFO, WARNING, and ERROR logs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>warning</strong>: WARNING and ERROR logs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>error</strong>: ERROR logs.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logging Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG</td>
<td>Detailed quantization processing information, including the quantization factors (<strong>scale</strong> and <strong>offset</strong>) and related debugging information.</td>
</tr>
<tr>
<td>INFO</td>
<td>Brief quantization processing information, including the quantized layer names and BN fusion information.</td>
</tr>
<tr>
<td>WARNING</td>
<td>Warning messages during quantization.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error messages during quantization.</td>
</tr>
</tbody>
</table>
The logging level is case insensitive. That is, **Info**, **info**, and **INFO** are all valid values.

**Returns**

None

**Example**

```python
import amct_tensorflow as amct
amct.set_logging_level(print_level="info", save_level="info")
amct.quantize_model(
    graph=tf.get_default_graph(),
    config_file="/configs/config.json",
    record_file="/record_scale_offset.txt")
...
amct.save_model(pb_model="/user_model.pb",
    outputs=['model/outputs'],
    record_file="/record_scale_offset.txt",
    save_path="/inference/",
    save_type="Both")
```

### 5.9.2 Post-training Quantization

#### 5.9.2.1 create_quant_config

**Description**

Applies to post-training quantization. Finds all quantization-capable layers in a graph, creates a quantization configuration file, and writes the quantization configuration information of the quantization-capable layers to the file.

**Prototype**

```python
create_quant_config(config_file, graph, skip_layers=None, batch_num=1, activation_offset=True, config_defination=None)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the quantization configuration file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The existing file (if available) in the directory will be overwritten upon this API call.</td>
<td></td>
</tr>
<tr>
<td>graph</td>
<td>Input</td>
<td>A <code>tf.Graph</code> of the model to be quantized.</td>
<td>A <code>tf.Graph</code>.</td>
</tr>
</tbody>
</table>

_CANN Development Auxiliary Tool Guide (Training) 5 AMCT Instructions (TensorFlow) Issue 01 (2021-07-01) Copyright © Huawei Technologies Co., Ltd._
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input / Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip_layers</td>
<td>Input</td>
<td>Quantizable layers in the <code>tf.Graph</code> to skip.</td>
<td>Default: <strong>None</strong>&lt;br&gt;A list of strings.&lt;br&gt;Restriction: If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect.</td>
</tr>
<tr>
<td>batch_num</td>
<td>Input</td>
<td>Number of batches taken to generate the quantization factors.</td>
<td>An int.&lt;br&gt;Value range: any integer larger than 0.&lt;br&gt;Default: <strong>1</strong>&lt;br&gt;Restrictions:&lt;br&gt;● <code>batch_num</code> cannot be too large. The product of <code>batch_num</code> and <code>batch_size</code> equals to the number of images used during quantization. Too many images consume too much memory.&lt;br&gt;● If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect.</td>
</tr>
<tr>
<td>activation_offset</td>
<td>Input</td>
<td>Whether to quantize activations with offset.</td>
<td>Default: <strong>true</strong>&lt;br&gt;A bool.&lt;br&gt;Restriction: If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>config_definition</td>
<td>Input</td>
<td>Whether to create a simplified quantization configuration file <strong>quant.cfg</strong> from the <strong>calibration_config_tf.proto</strong> file in /amct_tensorflow/proto/ <strong>calibration_config_tf.proto</strong> in the AMCT installation path. For details about the parameters in the <strong>calibration_config_tf.proto</strong> template and the generated simplified quantization configuration file <strong>quant.cfg</strong>, see 5.11.2 Simplified Post-training Quantization Configuration File.</td>
<td>Default: <strong>None</strong> A string. Restriction: If it is set to <strong>None</strong>, a configuration file is generated based on the residual arguments (<strong>skip_layers</strong>, <strong>batch_num</strong>, and <strong>activation_offset</strong>). Otherwise, a configuration file in JSON format is generated based on this argument.</td>
</tr>
</tbody>
</table>

**Returns**

None

**Outputs**

Outputs a quantization configuration file in JSON format. (When quantization is performed again, the existing one output by this API call will be overwritten.)

```json
{
    "version":1,
    "batch_num":1,
    "activation_offset":true,
    "joint_quant":false,
    "do_fusion":true,
    "skip_fusion_layers":[]
}
"MobilenetV2/Conv/Conv2D":{
    "quant_enable":true,
    "activation_quant_params":{
        "max_percentile":0.999999,
        "min_percentile":0.999999,
        "search_range":[0.7, 1.3],
        "search_step":0.01,
        "weight_quant_params":{
            "wts_algo":"arq_quantize",
            "channel_wise":true
        }
    },
    "MobilenetV2/Conv_1/Conv2D":{
    "quant_enable":true,
```
"activation_quant_params":{
  "max_percentile":0.999999,
  "min_percentile":0.999999,
  "search_range"[:
    0.7,
    1.3
  ],
  "search_step":0.01
},
"weight_quant_params":{
  "wts_algo":"arq_quantize",
  "channel_wise":true
}
"MobilenetV2/Logits/AvgPool":{
  "quant_enable":true,
  "activation_quant_params":{
    "max_percentile":0.999999,
    "min_percentile":0.999999,
    "search_range"[:
      0.7,
      1.3
    ],
    "search_step":0.01
  },
  "weight_quant_params":{
    "wts_algo":"arq_quantize",
    "channel_wise":false
  }
}

Example

```python
import amct_tensorflow as amct
# Create a graph of the network to be quantized.
network = build_network()
# Create a quantization configuration file.
amct.create_quant_config(config_file="./configs/config.json",
graph=tf.get_default_graph(),
skip_layers=None,
batch_num=1,
activation_offset=True)
```

5.9.2.2 quantize_model

Description

Applies to post-training quantization. Quantizes a graph based on the quantization configuration file, inserts the quantization operators, generates a quantization factor record file `record_file`, and returns the list of newly added operators.

Prototype

```python
quant_add_ops = quantize_model(graph, config_file, record_file)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph</td>
<td>Input</td>
<td>A tf.Graph of the model to be quantized.</td>
<td>A tf.Graph. Must be an inference graph containing no training-mode operators. For example, is_training of the FusedBatchNormV3 operator must be False.</td>
</tr>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization configuration file generated by the user, which is used to specify the configuration of the quantization layer in the tf.Graph graph of the model</td>
<td>A string.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td>quant_ad_d_ops</td>
<td>Return</td>
<td>List of operators to be inserted for quantization.</td>
<td>A list of tf.Variables.</td>
</tr>
</tbody>
</table>

Returns

A list of quantized layers on the network.

The 5.9.2.2 quantize_model call performs fusion on the graph, which might alter the output nodes. For example, Conv+BN (or Conv+BiasAdd+BN) is fused into Conv+BiasAdd, and an output node equivalent to BN is a BiasAdd node.

Example

```python
import amct_tensorflow as amct

# Create a network to be quantized.
network = build_network()

# Quantize the model.
amct.quantize_model(
    graph=tf.get_default_graph(),
    config_file="./configs/config.json",
    record_file="./record_scale_offset.txt")
```

5.9.2.3 save_model

Description

Applies to post-training quantization. Saves the original .pb model file to be quantized as a .pb model file that can be used for both accuracy simulation in the
TensorFlow environment and inference on the Ascend AI Processor based on the `record_file` quantization factor record file.

**Restrictions**

- This API is called after `batch_num` is reached. Otherwise, the quantization factors are incorrect and the quantization result is compromised.
- The model file passed to the API call must be in .pb format. You might need to convert your models to the .pb format in advance.
- The quantization factor record file passed to the API call is generated in the 5.9.2.2 `quantize_model` phase. The factor values will be filled in the model inference phase.

**Prototype**

```
save_model(pb_model, outputs, record_file, save_path)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb_model</td>
<td>Input</td>
<td>Original .pb model file to be quantized.</td>
<td>A string. Must be an inference graph containing no training-mode operators. For example, <code>is_training</code> of the FusedBatchNormV3 operator must be <code>False</code>.</td>
</tr>
<tr>
<td>outputs</td>
<td>Input</td>
<td>List of output operators of the graph.</td>
<td>A list of strings.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name. Generate a quantized model file based on the file, quantization configuration file, and original .pb model file.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, <code>./quantized_model/*model</code>.</td>
<td>A string.</td>
</tr>
</tbody>
</table>

**Returns**

None
Outputs

Outputs a .pb model file that can be used for both accuracy simulation in the TensorFlow environment and inference on the Ascend AI Processor.

When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.

Example

```python
import amct_tensorflow as amct

# Perform network inference and complete quantization during the inference.
for i in batch_num:
    sess.run(outputs, feed_dict={inputs: inputs_data})

# Insert the API call and save the quantized model as a .pb file.
    amct.save_model(pb_model=":/user_model.pb",
                   outputs=['model/outputs'],
                   record_file=":/record_scale_offset.txt",
                   save_path=":/inference/model")
```

5.9.2.4 convert_model

Description

Based on the computed quantization factors, converts a TensorFlow model to a model for both accuracy simulation in the TensorFlow environment and inference on the Ascend AI Processor.

Restrictions

- The user model must match the quantization factor record file. For example, if the "Conv+BN" composite is fused before computing the quantization factors of Conv, the "Conv+BN" composite in the TensorFlow model to be converted also needs to be fused in advance.
- The format and content of the quantization factor record file must comply with the AMCT requirements defined in 5.11.4 Quantization Factor Record File.
- The quantization-capable layers include: Conv2D, MatMul, DepthwiseConv2dNative (dilation = 1), Conv2DBackpropInput (dilation = 1), and AvgPool.
- This API supports the fusion of the "Conv+BN", "Depthwise_Conv+BN", and "Group_conv+BN" composites in the user model. Layer-level fusion switch is supported.
- Quantization of only an original floating-point model is supported. The model cannot be quantized if the input model contains any of the following custom quantization layers: QuantIfmr, QuantArq, SearchN, AscendQuant, AscendDequant, AscendAntiQuant, and AscendWeightQuant.

Prototype

```python
convert_model(pb_model, outputs, record_file, save_path)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb_model</td>
<td>Input</td>
<td>Original .pb model file to be quantized.</td>
<td>A string. Must be an inference graph containing no training-mode operators. For example, is_training of the FusedBatchNormV3 operator must be False.</td>
</tr>
<tr>
<td>outputs</td>
<td>Input</td>
<td>List of output operators of the graph.</td>
<td>A list.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Path of the quantization factor record file (.txt) computed by the user.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, ./quantized_model/<em>model</em>.</td>
<td>A string.</td>
</tr>
</tbody>
</table>

Returns

None

Outputs

Outputs a .pb model file that can be used for both accuracy simulation in the TensorFlow environment and inference on the Ascend AI Processor.

When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.

Example

```python
import amct_tensorflow as amct
convert_model(pb_model='./user_model.pb',
              outputs=['model/outputs'],
              record_file='./record_quantized.txt',
              save_path='./quantized_model/model')
```

5.9.2.5 accuracy_based_auto_calibration

Description

Performs automatic calibration on the input model based on the input configuration file to search for a quantization configuration that meets the accuracy requirement, resulting in a model that can serve both for accuracy.
simulation (**fake_quant**) in the TensorFlow environment and for inference (**deploy**) on the Ascend AI Processor.

**Restrictions**

None

**Prototype**

```
accuracy_based_auto_calibration(model_file, outputs, record_file, config_file, save_dir, evaluator, strategy='BinarySearch', sensitivity='CosineSimilarity')
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputs</td>
<td>Input</td>
<td>List of output nodes.</td>
<td>A string.</td>
</tr>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization configuration file.</td>
<td>A string.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Quantization factor record file. If the file exists, it will be overwritten.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_dir</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, ./quantized_model/*model.</td>
<td>A string.</td>
</tr>
<tr>
<td>evaluator</td>
<td>Input</td>
<td>Python instance for automatic quantization calibration and accuracy evaluation.</td>
<td>A Python instance.</td>
</tr>
<tr>
<td>strategy</td>
<td>Input</td>
<td>Strategy of searching for the quantization configuration that meets accuracy requirements. By default, the binary search strategy is used.</td>
<td>A string or a Python instance. Default: <strong>BinarySearch</strong></td>
</tr>
<tr>
<td>sensitivity</td>
<td>Input</td>
<td>Metric used to evaluate the quantization sensitivity of each quantization layer. By default, the cosine similarity metric is used.</td>
<td>A string or a Python instance. Default: <strong>CosineSimilarity</strong></td>
</tr>
</tbody>
</table>

**Returns**

None
Outputs

- Outputs a .pb model file that can be used for both accuracy simulation in the TensorFlow environment and inference on the Ascend AI Processor.
- Outputs the quantization factor record file, quantization configuration file, layer similarity result file, and automatic quantization rollback history file.

Example

```python
import amct_tensorflow as amct
from amct_tensorflow.common.auto_calibration.auto_calibration_evaluator_base import AutoCalibrationEvaluatorBase
from amct_tensorflow.accuracy_based_auto_calibration import accuracy_based_auto_calibration
from amct_tensorflow.common.auto_calibration.binary_search_strategy import BinarySearchStrategy
from amct_tensorflow.common.auto_calibration.cosine_similarity_sensitivity import CosineSimilaritySensitivity

def main():
    args_check(args)
    outputs = [PREDICTIONS]
    record_file = os.path.join(RESULT_DIR, 'record.txt')
    config_file = os.path.join(RESULT_DIR, 'config.json')
    with tf.io.gfile.GFile(args.model, mode='rb') as model:
        graph_def = tf.compat.v1.GraphDef()
        graph_def.ParseFromString(model.read())
        tf.import_graph_def(graph_def, name='')
        amct.create_quant_config(config_file, graph)
        save_dir = os.path.join(RESULT_DIR, 'MobileNetV2')
        evaluator = MobileNetV2Evaluator(args.dataset, args.keyword, args.num_parallel_reads, args.batch_size)
        accuracy_based_auto_calibration(args.model, outputs, record_file, config_file, save_dir, evaluator)
```

5.9.3 Quantization Aware Training

5.9.3.1 create_quant_retrain_config

Description

Applies to quantization aware training. Finds all quantization-capable layers in a graph, creates a quantization configuration file, and writes the quantization configuration information of the quantization-capable layers to the configuration file.

Restrictions

None

Prototype

```python
create_quant_retrain_config(config_file, graph, config_defination=None)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the configuration file, including the file name. The existing file (if available) in the directory will be overwritten upon this API call.</td>
<td>A string.</td>
</tr>
<tr>
<td>graph</td>
<td>Input</td>
<td>A <code>tf.Graph</code> of the model to be quantized.</td>
<td>A <code>tf.Graph</code></td>
</tr>
<tr>
<td>config_definition</td>
<td>Input</td>
<td>Simplified quantization configuration file <code>quantcfg</code>, which is generated from the <code>retrain_config_tf.proto</code> file. The <code>retrain_config_tf.proto</code> file is stored in <code>/amct_tensorflow/proto/retrain_config_tf.proto</code> in the AMCT installation path. For details about the parameters in the <code>retrain_config_tf.proto</code> template and the generated simplified quantization configuration file <code>quantcfg</code>, see 5.11.3 Simplified Quantization Aware Training Configuration File.</td>
<td>Default: None A string. Restriction: If it is set to None, a configuration file is generated based on the residual arguments. Otherwise, a configuration file in JSON format is generated based on this argument.</td>
</tr>
</tbody>
</table>

Returns

None

Outputs

Outputs a quantization aware training configuration file in JSON format. The existing configuration file (if available) in the directory will be overwritten upon this API call. An example is as follows:

```json
{
  "version":1,
  "batch_num":1,
  "conv":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo":"ulq_quantize"
    },
    "retrain_weight_config":{
      "algo":"arq_retrain",
      "channel_wise":true
    }
  }
}
```
Example

```python
import amct_tensorflow as amct
PATH, _ = os.path.split(os.path.realpath(__file__));
config_path = os.path.join(PATH, 'resnet50_config.json');
simple_config = './retrain/retrain.cfg';
graph = tf.compat.v1.get_default_graph();
amct.create_quant_retrain_config(config_path, graph, simple_config)
```

### 5.9.3.2 create_quant_retrain_model

**Description**

Applies to quantization aware training. Runs quantization aware training on a graph based on the `config_file` configuration file, inserts activations' and weights' fake-quantization layers, and saves the modified network to a new file.

**Restrictions**

None

**Prototype**

```python
retrain_ops = create_quant_retrain_model(graph, config_file, record_file)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph</td>
<td>Input</td>
<td>A <code>tf.Graph</code> of the model to be quantized.</td>
<td>A <code>tf.Graph</code>.</td>
</tr>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization aware training configuration file for the <code>tf.Graph</code> model.</td>
<td>A string.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>retrain_ops</td>
<td>Return</td>
<td>List of new variables for quantization aware training.</td>
<td>A list of <code>tf.Variables</code>.</td>
</tr>
</tbody>
</table>

**Returns**

List of new variables for quantization aware training.

**Outputs**

None

**Example**

```python
import amct_tensorflow as amct
retrain_ops = amct.create_quant_retrain_model(graph, config_path, record_file)
```

### 5.9.3.3 save_quant_retrain_model

**Description**

Applies to quantization aware training. Saves a retrained model to a model that can be used for both accuracy simulation and inference.

**Restrictions**

None

**Prototype**

```python
save_quant_retrain_model(pb_model, outputs, record_file, save_path)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb_model</td>
<td>Input</td>
<td>Retrained .pb model file.</td>
<td>A string. Must be an inference graph containing no training-mode operators. For example, <code>is_training</code> of the FusedBatchNormV3 operator must be <code>False</code>.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>outputs</td>
<td>Input</td>
<td>Model outputs.</td>
<td>A list of strings, for example, [output1,output2, ...].</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Quantization factor record file. It is used to generate a quantized model file.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, ./quantized_model/*model.</td>
<td>A string.</td>
</tr>
</tbody>
</table>

Returns

None

Outputs

Outputs a .pb model file that can be used for both accuracy simulation in the TensorFlow environment and inference on the Ascend AI Processor.

When quantization aware training is performed again, the existing files in the output directory will be overwritten upon this API call.

Example

```python
import amct_tensorflow as amct
amct.save_quant_retrain_model(FLAGS.checkpoint_path+'/output_graph.pb',output_node_names, 
FLAGS.checkpoint_path+'/resnet50')
```

5.9.4 Tensor Decomposition

5.9.4.1 auto_decomposition

Description

Generates a decomposed model file and its weight files from a given original TensorFlow model and its weight file.

Restrictions

- The input model file (.meta) must match the weight files (.data-0000X-of-0000X and .index). The .data-0000X-of-0000X weight file must match the .index weight index file.
- Pass the directory of the original model and an output directory to the call to the tensor decomposition API. The API automatically decomposes any Convolution layer that meets the decomposition conditions. For details about the decomposition conditions, see Restrictions.
Prototype

```python
add_ops = auto_decomposition(meta_path, ckpt_path, save_path)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>meta_path</td>
<td>Input</td>
<td>TensorFlow model definition file (.meta).</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>NOTE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ensure that the .meta file can be properly loaded by <code>tf.compat.v1.train.import_meta_graph</code>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example, for a .meta file obtained through Horovod training, you must run the <code>import hovorod.tensorflow</code> command for the file to be successfully called by 5.9.4.1 <code>auto_decomposition</code>.</td>
<td></td>
</tr>
<tr>
<td>ckpt_path</td>
<td>Input</td>
<td>Weight files of the already-trained TensorFlow model (.data-0000X-of-0000X and .index). You only need to specify the common prefix of the .data-0000X-of-0000X file and .index file.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Directory of the result files.</td>
<td>A string.</td>
</tr>
<tr>
<td>add_ops</td>
<td>Return</td>
<td>List of newly added convolutional layers after model decomposition.</td>
<td>A list.</td>
</tr>
</tbody>
</table>

Returns

List of newly added convolutional layers after model decomposition.

Outputs

- Checkpoint file when saving the TensorFlow model.
- .meta model file after tensor decomposition.
- .data-00000-of-00001 weight file after tensor decomposition.
- .index weight file index after tensor decomposition.
- .pkl file that records the graph structure changes during tensor decomposition, which will be passed to the 5.9.4.2 `decompose_graph` call.

Example

```python
from amct_caffe.tensor_decompose import auto_decomposition
auto_decomposition(meta_path='model.ckpt-2000.meta',
                    ckpt_path='model.ckpt-200',
                    save_path='model_decomposed')
```
5.9.4.2 decompose_graph

Description
Decomposes a graph in the decomposition training code for fine-tuning the decomposed model.

Restrictions
- **5.9.4.1 auto_decomposition** have been called to decompose the model.
- This API must be used based on the training code, and the model and weight decomposed by the **5.9.4.1 auto_decomposition** call must be obtained based on the training code.
- This API call modifies only the graph, but does not modify the references to the original variables. If the decomposed convolution is the last node in the graph, the decomposition of the convolution may be invalid.

Prototype

```python
add_ops = decompose_graph(save_path, graph=None)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Directory of the result files generated by the <strong>5.9.4.1 auto_decomposition</strong> call.</td>
<td>A string.</td>
</tr>
<tr>
<td>graph</td>
<td>Input</td>
<td>(Optional) Graph to decompose. If it is not specified, the graph of the current model is used. Defaults to <strong>None</strong>.</td>
<td>A <strong>tf.Graph</strong>.</td>
</tr>
<tr>
<td>add_ops</td>
<td>Return</td>
<td>List of newly added convolutional layers after model decomposition.</td>
<td>A list.</td>
</tr>
</tbody>
</table>

Returns
List of newly added convolutional layers after model decomposition.

Outputs
None

Example

```python
from amct_tensorflow.tensor_decompose import decompose_graph
decompose_graph(save_path)
```
5.9.5 QAT Model to Ascend Model Conversion

5.9.5.1 convert_qat_model

Description
Converts a TensorFlow QAT model to a quantized model that serves for both deployment on the Ascend AI Processor and accuracy simulation on CPU and GPU.

Restrictions
The source TensorFlow .pb model must have FakeQuantWithMinMaxVars and FakeQuantWithMinMaxVarsPerchannel operators.

Prototype
convert_qat_model(pb_model, outputs, save_path, record_file=None)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb_model</td>
<td>Input</td>
<td>Path of the QAT model to convert.</td>
<td>A string. Must be an inference graph containing no training-mode operators.</td>
</tr>
<tr>
<td>outputs</td>
<td>Input</td>
<td>List of output operators of the graph.</td>
<td>A list.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Path of the quantization factor record file (.txt) computed by the user.</td>
<td>A string. Default: None</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, ./quantized_model/<em>model</em></td>
<td>A string.</td>
</tr>
</tbody>
</table>

Returns
List of output operators of the graph.

Outputs
Outputs a .pb model file that can be used for both accuracy simulation in the TensorFlow environment and inference on the Ascend AI Processor.
When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.

Example

```python
import amct_tensorflow as amct
convert_qat_model(pb_model, outputs, save_path)
```

5.10 FAQs

5.10.1 An Error Message Is Displayed During python3-tk Installation

Symptom

When the python3-tk dependency is installed, the following error message is displayed.

```bash
python3-tk is already the newest version (3.6.8-1-18.04).
The following packages were automatically installed and are no longer required:
aptitude-common libboost-iostreams1.58.0 libcairo-3-5 libgcr4-2.0-0v5 libx11-62v5
0 upgraded, 0 newly installed, 0 to remove and 304 not upgraded.
1 not fully installed or removed.
After this operation, 0.8 of additional disk space will be used.
Do you want to continue? [y/n] Y
```

Solution

Copy the missing file `py_compile.py` to the `/usr/lib/python3.7` directory and reinstall the Python.

```bash
cp /usr/local/python3.7.5/lib/python3.7/py_compile.py /usr/lib/python3.7
```

Replace `/usr/local/python3.7.5/lib/python3.7/py_compile.py` with the actual path of the file.

5.10.2 AMCT Resource Allocation Failure

Symptom

- Scenario 1: The size of accumulated data is too large and overflows.
  The failure information is as follows:
  ```python
tensorflow.python.framework.errors_impl.InvalidArgumentError: Check failed: size <= tensorflow::kint32max (2684354560 vs. 2147483647)
```
  - Scenario 2: The memory or video RAM overflows.
    When AMCT is used, some extra memory or video RAM space is allocated. If the memory or video RAM resources are insufficient, allocation fails. As a result, the system displays a message indicating that resource allocation fails.
The common system resource allocation failure information is as follows:

In the CPU operating environment, part of the memory allocation failure information is as follows:

```
MemoryError: Unable to allocate array with shape (1048576, 102400) and data type float32
```

In the GPU operating environment, part of the video RAM allocation failure information is as follows:

```
ResourceExhaustedError (see above for traceback): OOM when allocating tensor with shape [1, 1073741824] and type int32 on /job:localhost/replica:0/task:0/device:GPU:0 by allocator G
[[node TopKV2_1 (defined at test_input_big_tensor.py:30) = TopKV2[T=DT_FLOAT, sorted=true, _device="/job:localhost/replica:0/task:0/device:GPU:0"](Reshape, Cast_1/_5)]]
```

Hint: If you want to see a list of allocated tensors when OOM happens, add `report_tensor_allocations_upon_oom` to RunOptions for current allocation info.

```
[[node QuantIfmr/_17] = _Recv[client_terminated=false, recv_device="/job:localhost/replica:0/task:0/device:CPU:0", send_device="/job:localhost/replica:0/task:0/device:arnation=1, tensor_name="edge_55_QuantIfmr", tensor_type=DT_FLOAT, _device="/job:localhost/replica:0/task:0/device:CPU:0"]()]
```

Hint: If you want to see a list of allocated tensors when OOM happens, add `report_tensor_allocations_upon_oom` to RunOptions for current allocation info.

**Solution**

You are advised to decrease the value of `batch_num` for quantization. Depending on the hardware resources, resource allocation may still fail after the value of `batch_num` is decreased. You are advised to decrease the `batch_num` argument and batch size, or use a platform with sufficient hardware resources for quantization.

**5.11 Appendixes**

**5.11.1 Sample Directory Description**

Decompress the `amct_tensorflow_sample.tar.gz` package. The extracted `amct_tf/sample` directory is organized as follows.

<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mobilenetv2</td>
<td>-</td>
<td>Image classification directory.</td>
</tr>
<tr>
<td></td>
<td>calibration</td>
<td>Calibration dataset, containing 32 images.</td>
</tr>
<tr>
<td></td>
<td>calibration_reference.txt</td>
<td>Link to images in the calibration dataset.</td>
</tr>
<tr>
<td></td>
<td>classification.jpg</td>
<td>Image dataset.</td>
</tr>
<tr>
<td></td>
<td>convert/record_quantized.txt</td>
<td>File that records quantization factors of the original MobileNetV2 model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accuracy-oriented automatic quantization rollback script.</td>
</tr>
<tr>
<td>Level-1 Directory</td>
<td>Level-2 Directory</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>MobileNetV2_convert_model.py</td>
<td>Quantization script using the convert_model API.</td>
</tr>
<tr>
<td></td>
<td>MobileNetV2_sample.py</td>
<td>Quantization script.</td>
</tr>
<tr>
<td></td>
<td>pre_model/mobilenet_v2.pb</td>
<td>Original MobileNetV2 model file.</td>
</tr>
<tr>
<td>yolov3</td>
<td>-</td>
<td>Object detection directory.</td>
</tr>
<tr>
<td></td>
<td>calibration.jpg</td>
<td>Calibration image.</td>
</tr>
<tr>
<td></td>
<td>COCO_labels.txt</td>
<td>Calibration dataset label file.</td>
</tr>
<tr>
<td></td>
<td>detection.jpg</td>
<td>Image dataset.</td>
</tr>
<tr>
<td></td>
<td>pre_mode/yolov3_coco.pb</td>
<td>Model file.</td>
</tr>
<tr>
<td></td>
<td>yolo_quant.cfg</td>
<td>Simplified quantization configuration file.</td>
</tr>
<tr>
<td></td>
<td>YOLOV3_sample.py</td>
<td>Quantization script.</td>
</tr>
<tr>
<td>resnet_v1_50</td>
<td>-</td>
<td>Model directory for quantization aware training.</td>
</tr>
<tr>
<td></td>
<td>pre_model</td>
<td>Model directory, containing:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● resnet_v1_50_eval.meta: model evaluation file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● resnet_v1_50.index: weight file index.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● resnet_v1_50_train.meta: model file for quantization aware training.</td>
</tr>
<tr>
<td></td>
<td>resnet_v1_50_retrain_sample.py</td>
<td>Quantization aware training script.</td>
</tr>
<tr>
<td></td>
<td>sample.cfg</td>
<td>Simplified quantization aware training configuration file.</td>
</tr>
<tr>
<td>tensor_decompose</td>
<td>-</td>
<td>Tensor decomposition directory.</td>
</tr>
<tr>
<td></td>
<td>decompose_sample.py</td>
<td>Tensor decomposition script.</td>
</tr>
<tr>
<td></td>
<td>finetune_sample_estimator.py</td>
<td>Estimator fine-tuning script.</td>
</tr>
<tr>
<td></td>
<td>finetune_sample_session.py</td>
<td>Session fine-tuning script.</td>
</tr>
<tr>
<td></td>
<td>README.md</td>
<td>Tensor decomposition readme file.</td>
</tr>
</tbody>
</table>
Level-1 Directory | Level-2 Directory | Description
--- | --- | ---
 | train_sample_estimator.py | Estimator training script.
 | train_sample_session.py | Session training script.

### 5.11.2 Simplified Post-training Quantization Configuration File

Table 5-39 describes the parameters in the `calibration_config_tf.proto` template.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified post-training quantization configuration file of AMCT.</td>
</tr>
<tr>
<td>Optional</td>
<td>uint32</td>
<td>batch_num</td>
<td>Batch count used for quantization.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>activation_offset</td>
<td>Whether to quantize activations with offset.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>joint_quant</td>
<td>Eltwise joint quantization switch. Defaults to false, indicating that joint quantization is disabled. If it is set to true, the network performance may improve but the accuracy may be compromised.</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td>-----------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_layers</td>
<td>Layers to skip during quantization.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_layer_types</td>
<td>Types of layers to skip during quantization.</td>
</tr>
<tr>
<td>Optional</td>
<td>NuqConfig</td>
<td>nuq_config</td>
<td></td>
<td>Non-uniform quantization configuration. This version does not support non-uniform quantization.</td>
</tr>
<tr>
<td></td>
<td>CalibratorConfig</td>
<td>common_config</td>
<td></td>
<td>Common quantization configuration. If a layer is not overridden by override_layer_types or override_layer_configs, this configuration is used.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>Override LayerType</td>
<td>override_layer_types</td>
<td>Type of layers to override.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>Override Layer</td>
<td>override_layer_configs</td>
<td>Layer to override.</td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>do_fusion</td>
<td></td>
<td>BN fusion switch. Defaults to true.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_fusion_layers</td>
<td>Layers to skip in BN fusion.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------</td>
<td>-------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NuqConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Non-uniform quantization configuration. This version does not support non-uniform quantization.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>mapping_file</td>
<td>JSON file converted from the offline model converted using the ATC tool from the deployable model generated form uniform quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>NUQuan</td>
<td>nuq_quantize</td>
<td>Non-uniform quantization configuration.</td>
</tr>
<tr>
<td>OverrideLayerType</td>
<td>Required</td>
<td>string</td>
<td>layer_type</td>
<td>Type of layers to quantize.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>calibration_config</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td>OverrideLayer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization configuration overriding by layer.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>layer_name</td>
<td>Layer to override.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>calibration_config</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td>CalibrationConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Calibration-based quantization configuration.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ARQuantize</td>
<td>-</td>
<td>arq_quantize</td>
<td></td>
<td>Weight quantization algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>arq_quantize</td>
<td>arq_quantize: ARQ algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>nuq_quantize</td>
<td></td>
<td>Weight quantization algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>nuq_quantize</td>
<td>nuq_quantize: non-uniform quantization algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ifmr_quantize</td>
<td>ifmr_quantize</td>
<td>Activation quantization algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ifmr_quantize: IFMR algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARQuantize</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ARQ quantization algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bool</td>
<td>channel_wise</td>
<td>Whether to use different quantization factors for each channel.</td>
</tr>
<tr>
<td>FMRQuantize</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>FMR quantization algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>float</td>
<td>search_range_start</td>
<td>Quantization factor search start.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>float</td>
<td>search_range_end</td>
<td>Quantization factor search end.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>float</td>
<td>search_step</td>
<td>Quantization factor search step.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>max_percentile</td>
<td>Upper search limit.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>min_percentile</td>
<td>Lower search limit.</td>
</tr>
<tr>
<td>NUQuantize</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Non-uniform quantization configuration. This version does not support non-uniform quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>uint32</td>
<td>num_steps</td>
<td>Number of steps for non-uniform quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>uint32</td>
<td>num_of_iteration</td>
<td>Number of iterations for non-uniform quantization optimization.</td>
</tr>
</tbody>
</table>

The following is an example simplified uniform quantization configuration file (`quant.cfg`).

```
# Global quantization parameters
batch_num : 2
activation_offset : true
joint_quant : false
skip_layers : "conv_1"
skip_layer_types:"Conv2D"
do_fusion: true
skip_fusion_layers : "conv_1"
common_config : {
    arq_quantize : {
        channel_wise : true
    } ifmr_quantize : {
        search_range_start : 0.7
        search_range_end : 1.3
        search_step : 0.01
        max_percentile : 0.999999
        min_percentile : 0.999999
    }
}
override_layer_types : {
    layer_type : "MatMul"
}
```
5.11.3 Simplified Quantization Aware Training Configuration File

Table 5-40 describes the parameters in the retrain_config_tf.proto template.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTRetrainConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified quantization aware training configuration file of AMCT.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>uint32</td>
<td>batch_num</td>
<td>Batch count used for quantization.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>RetrainDataQuantConfig</td>
<td>retrain_data_quant_config</td>
<td>Activation retrain configuration.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>RetrainWeightQuantConfig</td>
<td>retrain_weight_quant_config</td>
<td>Weight retrain configuration.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_layers</td>
<td>Layers to skip.</td>
</tr>
</tbody>
</table>

Table 5-40 Parameter description
<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RetrainDataQuantConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Activation retrain configuration.</td>
</tr>
<tr>
<td>-</td>
<td>ULQuantize</td>
<td>ulq_quantize</td>
<td></td>
<td>Activation quantization algorithm. Only ULQ is supported in the current version.</td>
</tr>
<tr>
<td>ULQuantize</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ULQ algorithm configuration.</td>
</tr>
<tr>
<td>Optional</td>
<td>ClipMaxMin</td>
<td>clip_max_min</td>
<td></td>
<td>Initial upper and lower limits. If it is not specified, IFMR is used for initialization.</td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>fixed_min</td>
<td></td>
<td>Whether to fix the lower limit at 0. Set to true for ReLU or false for other algorithms.</td>
</tr>
<tr>
<td>ClipMaxMin</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Initial upper and lower limits.</td>
</tr>
<tr>
<td>Required</td>
<td>float</td>
<td>clip_max</td>
<td></td>
<td>Initial upper limit.</td>
</tr>
<tr>
<td>Required</td>
<td>float</td>
<td>clip_min</td>
<td></td>
<td>Initial lower limit.</td>
</tr>
<tr>
<td>RetrainWeightQuantConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Weight retrain configuration.</td>
</tr>
<tr>
<td>-</td>
<td>ARQRetrain</td>
<td>arq_retrain</td>
<td></td>
<td>Weight quantization algorithm. Only ARQ is supported in the current version.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------</td>
<td>----------</td>
<td>-------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>ARQRetrain</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ARQ algorithm configuration.</td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>channel_wise</td>
<td>Channel-wise ARQ enable.</td>
<td></td>
</tr>
<tr>
<td>RetrainOverride-Layer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Layer overriding configuration.</td>
</tr>
<tr>
<td>Required</td>
<td>string</td>
<td>layer_name</td>
<td>Layer name.</td>
<td></td>
</tr>
<tr>
<td>Required</td>
<td>RetrainDataQuantConfig</td>
<td>retrain_data_quant_config</td>
<td>Activation quantization configuration to apply.</td>
<td></td>
</tr>
<tr>
<td>Required</td>
<td>RetrainWeightQuantConfig</td>
<td>retrain_weight_quant_config</td>
<td>Weight quantization configuration to apply.</td>
<td></td>
</tr>
<tr>
<td>RetrainOverrideLayerType</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Types of layers to override.</td>
</tr>
<tr>
<td>Required</td>
<td>string</td>
<td>layer_type</td>
<td>Layer type.</td>
<td></td>
</tr>
<tr>
<td>Required</td>
<td>RetrainDataQuantConfig</td>
<td>retrain_data_quant_config</td>
<td>Activation quantization configuration to apply.</td>
<td></td>
</tr>
<tr>
<td>Required</td>
<td>RetrainWeightQuantConfig</td>
<td>retrain_weight_quant_config</td>
<td>Weight quantization configuration to apply.</td>
<td></td>
</tr>
</tbody>
</table>

The following is an example simplified quantization aware training configuration file (*quant.cfg*).

```
# Global quantization parameters
retrain_data_quant_config: {
    ulq_quantize: {
        clip_max_min: {
            clip_max: 6.0
            clip_min: -6.0
        }
    }
}

retrain_weight_quant_config: {
    arq_retrain: {
        channel_wise: true
    }
}
```
5.11.4 Quantization Factor Record File

Prototype

The quantization factor record file is a serialized data structure file based on Protobuf. You can generate a quantized model file by using the quantization configuration file, original network model file, and the quantization factor record file.

- Protobuf prototype for quantization with 5.9.2.4 convert_model (find the code in the /amct_tensorflow/proto/scale_offset_record_tf.proto file under the AMCT installation directory)
  
  ```protobuf
  syntax = "proto2";
  package AMCTTensorflow;
  // this proto is designed for convert_model API
  message SingleLayerRecord {
    optional float scale_d = 1;
    optional int32 offset_d = 2;
    repeated float scale_w = 3;
    repeated int32 offset_w = 4;
    // convert_model does not support this field [shift_bit] yet
    repeated uint32 shift_bit = 5;
    optional bool skip_fusion = 9 [default = false];
  }

  message MapFiledEntry {
    optional string key = 1;
    optional SingleLayerRecord value = 2;
  }

  message ScaleOffsetRecord {
    repeated MapFiledEntry record = 1;
  }
  ```

The parameters in this scenario are described as follows.
<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleLayer Record</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization factors for quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>scale_d</td>
<td>Scale factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>int32</td>
<td>offset_d</td>
<td>Offset factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td>Repeated</td>
<td>float</td>
<td>scale_w</td>
<td></td>
<td>Scale factor for weight quantization. Two quantization modes are supported: scalar (quantizing the weight of the current layer in a unified manner) and vector (quantizing the weight of the current layer channel-wise). The channel-wise quantization mode applies only to the Conv2D, DepthwiseConv2dNative, and Conv2DBackpropInput layers.</td>
</tr>
<tr>
<td>Repeated</td>
<td>int32</td>
<td>offset_w</td>
<td></td>
<td>Offset factor for weight quantization. Similar to scale_w, it also supports the scalar and vector quantization modes and the dimension configuration must be the same as that of scale_w. Currently weight quantization with offset is not supported, and offset_w must be 0.</td>
</tr>
<tr>
<td>Repeated</td>
<td>uint32</td>
<td>shift_bit</td>
<td></td>
<td>Shift factor. Reserved for the 5.9.2.4 convert_model API.</td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>skip_fusion</td>
<td></td>
<td>Whether to skip Conv+BN fusion, Depthwise_Conv+BN fusion, Group_conv+BN fusion, and BatchNorm fusion at the current layer. Defaults to false, indicating that fusion of the preceding types is performed.</td>
</tr>
<tr>
<td>ScaleOffset Record</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Map structure. To ensure compatibility, the discrete map structure is used.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------</td>
<td>---------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MapFiledEntry</td>
<td>Optional</td>
<td>string</td>
<td>key</td>
<td>Layer name.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>SingleLayerRecord</td>
<td>value</td>
<td>Quantization factor configuration.</td>
</tr>
</tbody>
</table>

- Protobuf prototype for post-training quantization or quantization aware training (find the code in the `/amct_tensorflow/proto/inner_scale_offset_record.proto` file under the AMCT installation directory)

```proto2
syntax = "proto2";
package AMCTTensorflow;

// this proto is designed for amct tools
message InnerSingleLayerRecord {
  optional float scale_d = 1;
  optional int32 offset_d = 2;
  repeated float scale_w = 3;
  repeated int32 offset_w = 4;
  repeated uint32 shift_bit = 5;
  // the cluster of nuq, only nuq layer has this field;
  repeated int32 cluster = 6;
  optional bool skip_fusion = 9 [default = false];
}

message InnerMapFiledEntry {
  optional string key = 1;
  optional InnerSingleLayerRecord value = 2;
}

message InnerScaleOffsetRecord {
  repeated InnerMapFiledEntry record = 1;
}
```

The parameters in this scenario are described as follows.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InnerSingleLayerRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization factors for quantization.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td>--------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>scale_d</td>
<td>Scale factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>int32</td>
<td>offset_d</td>
<td>Offset factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>float</td>
<td>scale_w</td>
<td>Scale factor for weight quantization. Two quantization modes are supported: scalar (quantizing the weight of the current layer in a unified manner) and vector (quantizing the weight of the current layer channel-wise). The channel-wise quantization mode applies only to the Conv2D, DepthwiseConv2dNative, and Conv2DBackpropInput layers.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>int32</td>
<td>offset_w</td>
<td>Offset factor for weight quantization. Similar to scale_w, it also supports the scalar and vector quantization modes and the dimension configuration must be the same as that of scale_w. Currently weight quantization with offset is not supported, and offset_w must be 0.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>uint32</td>
<td>shift_bit</td>
<td>Shift factor.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>int32</td>
<td>cluster</td>
<td>Cluster center. Valid only for non-uniform quantization. Not supported currently.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------</td>
<td>---------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>skip_fusion</td>
<td>Whether to skip Conv+BN fusion, Depthwise_Conv+BN fusion, Group_conv+BN fusion, and BatchNorm fusion at the current layer. Defaults to false, indicating that fusion of the preceding types is performed.</td>
</tr>
<tr>
<td>InnerScaleOffsetRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Map structure. To ensure compatibility, the discrete map structure is used.</td>
</tr>
</tbody>
</table>
| InnerMapFiledEntry       | Repeated          | Inner MapFiledEntry | record    | Each records a quantization factor of a quantization layer and consists of two members:  
  ● key: layer name.  
  ● value: quantization factor defined by SingleLayerRecord.                                                  |
| InnerMapFiledEntry       | Optional          | string  | key           | Layer name.                                                                                                                                 |
|                         | Optional          | Inner SingleLayerRecord | value   | Quantization factor configuration.                                                                                                           |

Beware that the Protobuf protocol does not report an error for repeated settings of optional fields. Instead, the most recent settings are used.

For general quantization layers, the following parameters need to be configured: `scale_d, offset_d, scale_w, offset_w, and shift_bit`. The `scale_w` and `offset_w` parameters are unavailable for AvgPool since the layer has no weight. The quantization factor record file corresponding to `inner_scale_offset_record.proto` is provided as an example below.

```protobuf
case "fc4/Tensordot/MatMul"
  {  
    key: "fc4/Tensordot/MatMul"  
    value {  
      scale_d: 0.0798481479  
      offset_d: 1  
      scale_w: 0.00297622895  
      offset_w: 0  
      shift_bit: 1  
    }  
  }  
  {  
    key: "depthwise"  
  }
```
Quantization Factors

The scale (a floating-point number) and offset quantization factors need to be provided for data and weight quantization. AMCT uses a unified quantization data structure. See the following expression.

\[
data_{\text{int}8} = \text{clip}_{\text{int}8}\left(\text{round}\left(\frac{\text{data}_{\text{float}}}{\text{scale}}\right) + \text{offset}\right)
\]

The value ranges are as follows:

- \( \text{scale} \in \left[\frac{1}{\text{FLT\_EPSILON}}, \frac{1}{\text{FLT\_EPSILON}}\right] \), \( \text{FLT\_EPSILON} \approx 1.1920929 \times 10^{-7} \)
- \( \text{offset} \in [-128, 127] \)

Quantization algorithms are classified into symmetric quantization algorithms and asymmetric quantization algorithms.

1. **Symmetric quantization algorithm:**
   
   The original high-precision data and quantized int8 data are converted into \( \text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int}8} \), where \( \text{scale} \) is a float32. To indicate positive and negative numbers, the signed int8 data type is used for \( \text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int}8} \). The following describes how to convert the original data into the int8 format. \( \text{round} \) is a rounding function. The value to be determined by the quantization algorithm is the constant \( \text{scale} \).

\[
data_{\text{int}8} = \text{round}\left(\frac{\text{data}_{\text{float}}}{\text{scale}}\right)
\]
Quantization of the weights and activations may be summarized as a process of searching for a scale. Because \( \text{data}_{\text{int8}} \) is a signed number, to ensure symmetry of the ranges represented by positive and negative values, an absolute value operation is first performed on all data, so that the range of the to-be-quantized data is changed to \( \text{data}_{\text{int8}} \), and then scale is determined. The range of positive int8 values is \([0, 127]\). Therefore, scale can be computed as follows:

\[
\text{scale} = \frac{\text{data}_{\text{max}}}{127}
\]

Therefore, the range of the int8 values is \([-128 \times \text{scale}, 127 \times \text{scale}]\). Data beyond the range \([-128 \times \text{scale}, 127 \times \text{scale}]\) is saturated to a boundary value, and then the quantization operation shown in the formula is performed.

2. Asymmetric quantization algorithm:

The difference from symmetric quantization algorithms lies in the data conversion mode. The scale and offset constants also need to be determined.

\[
\text{data}_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset})
\]

The uint8 data is converted through calculation based on the original high-accuracy data, which is shown in the following formula:

\[
\text{data}_{\text{uint8}} = \text{round} \left( \frac{\text{data}_{\text{float}}}{\text{scale}} - \text{offset} \right)
\]

\( \text{scale} \) is an fp32, \( \text{data}_{\text{uint8}} \) is an unsigned int8, and \( \text{offset} \) is an int8. The data range is \([\text{scale} \times \text{offset}, \text{scale} \times (255 + \text{offset})]\) if a value range of the to-be-quantized data is \([\text{data}_{\text{min}}, \text{data}_{\text{max}}]\). \( \text{scale} \) and \( \text{offset} \) are computed as follows:

\[
\text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}, \quad \text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}
\]

Unified quantization data format: By performing simple data conversion of the asymmetric quantization algorithm formula, the quantized data and the symmetric quantization algorithm are of the same type, that is, int. The specific conversion process is as follows:

int8 quantization is used as an example for description. The input original floating-point data is \( \text{data}_{\text{float}} \), the original quantized fixed-point number is \( \text{data}_{\text{float}} \), the quantization scale is \( \text{scale} \), and the quantization offset is \( \text{data}_{\text{float}} \) (the algorithm requires zero crossing to avoid too much accuracy drop). The calculation principle of quantization is as follows:

\[
\text{data}_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset}) = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset} + 128) = \text{scale} \times (\text{data}_{\text{uint8}} - \text{offset} \quad \text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255})
\]

Where,

\[
\text{data}_{\text{uint8}} = \text{data}_{\text{uint8}} - 128 \in [-128, 127], \text{offset} = -(\text{offset} + 128) \in [-128, 127]
\]

Through the foregoing conversion, the data may also be converted into the int8 format. After \( \text{scale} \) and the converted \( \text{offset} \) are determined, the int8 data converted from the original floating-point data is as follows:
5.11.5 Tensor Decomposition Specification Reference of Open-Source Networks

The accuracy specification is top1 ACC(%) for an image classification network, mAP(%) for an object detection network, or DSC(%) for an image segmentation network. The fine-tuning learning rate decreases from 0.1 times of the original learning rate.

<table>
<thead>
<tr>
<th>Network</th>
<th>Job Type</th>
<th>Dataset</th>
<th>Baseline Accuracy</th>
<th>Accuracy After Decomposition</th>
<th>Accuracy After Post-decomposition Fine-tuning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>Classification</td>
<td>ImageNet</td>
<td>70.66</td>
<td>44.02</td>
<td>70.34</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>Classification</td>
<td>ImageNet</td>
<td>74.2</td>
<td>54.92</td>
<td>74.15</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>Classification</td>
<td>ImageNet</td>
<td>75.6</td>
<td>73.64</td>
<td>75.91</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>Classification</td>
<td>ImageNet</td>
<td>78.52</td>
<td>76.97</td>
<td>78.24</td>
</tr>
<tr>
<td>Inception-v3</td>
<td>Classification</td>
<td>ImageNet</td>
<td>77.98</td>
<td>76.95</td>
<td>77.78</td>
</tr>
<tr>
<td>SSD</td>
<td>Detection</td>
<td>COCO 2017</td>
<td>27.2</td>
<td>24.2</td>
<td>27.9</td>
</tr>
<tr>
<td>Faster R-CNN</td>
<td>Detection</td>
<td>COCO 2017</td>
<td>32.5</td>
<td>31</td>
<td>32.2</td>
</tr>
<tr>
<td>Mask R-CNN</td>
<td>Detection</td>
<td>COCO 2017</td>
<td>37.9</td>
<td>36.8</td>
<td>38</td>
</tr>
<tr>
<td>UNet</td>
<td>Segmentation</td>
<td>SSTEM</td>
<td>87.63</td>
<td>85.05</td>
<td>87.57</td>
</tr>
</tbody>
</table>

5.11.6 What Do I Do If My TensorFlow Network Output Node Is Changed by AMCT?

**Symptom**

When AMCT calls the `quantize_model` API to modify the original TensorFlow graph, the output node at the bottom layer changes because a searchN layer has been inserted. In the quantization script, you need to replace the output node for
inference with the new output node after graph modification as prompted. The AMCT quantization log provides the original output node name and new output node name after graph modification.

Consider the following scenarios about the output node change due to graph modification:

- **Scenario 1:** The bottom layer of the network model is ADD or ADDV2, and the data to add is one-dimensional, which meets the bias-add condition.

**Figure 5-10** ADD or ADDV2 as the bottom layer

- If the bottom layer is Add, messages similar to the following are displayed during graph modification:

```
2020-09-01 09:31:04,896 - WARNING - [AMCT]:[replace_add_pass]: Replace ADD at the end of the network! You need to replace the old output node by the new output node in inference process!
<<<<<<<the name of the old output node is 'Add:0'

2020-09-01 09:31:04,979 - WARNING - [AMCT]:[quantize_model]: Insert searchN operator at the end of the network! You need to replace the old output node by the new output node in inference process!
>>>>>>>the name of the new output node is 'bias_add/BiasAdd:0'
```

- If the bottom layer is AddV2, messages similar to the following are displayed during graph modification:

```
2020-09-01 09:32:42,281 - WARNING - [AMCT]:[replace_add_pass]: Replace ADD at the end of the network! You need to replace the old output node by the new output node in inference process!
>>>>>>>the name of the old output node is 'add:0'

2020-09-01 09:32:42,362 - WARNING - [AMCT]:[quantize_model]: Insert searchN operator at the end of the network! You need to replace the old output node by the new output node in inference process!
>>>>>>>the name of the new output node is 'bias_add/BiasAdd:0'
```

---

CANN
Development Auxiliary Tool Guide (Training)
5 AMCT Instructions (TensorFlow)

Issue 01 (2021-07-01) Copyright © Huawei Technologies Co., Ltd. 273
Scenario 2: The bottom layer of the network is BiasAdd, and its upstream layer is Conv2D, DepthwiseConv2dNative, Conv2DBackpropInput, or MatMul.

**Figure 5-11** BiasAdd as the bottom layer, whose upstream layer is Conv2D

Messages similar to the following are displayed during graph modification:

2020-09-01 09:39:26,130 - WARNING - [AMCT]:[quantize_model]: Insert searchN operator at the end of the network! You need to replace the old output node by the new output node in inference process!

Scenario 3: The bottom layer of the network is Conv2D, DepthwiseConv2dNative, Conv2DBackpropInput, MatMul, or AvgPool.
Figure 5-12 Conv2D as the bottom layer

Messages similar to the following are displayed during graph modification:

2020-09-01 09:40:28.717 - WARNING - [AMCT]-[quantize_model]: Insert searchN operator at the end of the network! You need to replace the old output node by the new output node in inference process!

The name of the old output node is 'Conv2D:0' //Original output node
The name of the new output node is 'search_n_quant/search_n_quant_SEARCHN/Identity:0' //New output node

- Scenario 4: The bottom layer of the network is FusedBatchNorm, FusedBatchNormV2, or FusedBatchNormV3, and its upstream layer is Conv2D +(BiasAdd) or DepthwiseConv2dNative+(BiasAdd).
Figure 5-13 FusedBatchNormV3 as the bottom layer

Messages similar to the following are displayed during graph modification:

2020-09-01 09:44:08,637 - WARNING - [AMCT]:[conv_bn_fusion_pass]: Fused BN at the end of the network! You need to replace the old output node by the new output node in inference process!

The name of the old output node is 'batch_normalization/FusedBatchNormV3:0'

The name of the new output node is 'bias_add:0'

2020-09-01 09:44:08,717 - WARNING - [AMCT]:[quantize_model]: Insert searchN operator at the end of the network! You need to replace the old output node by the new output node in inference process!

The name of the old output node is 'bias_add:0'

The name of the new output node is 'search_n_quant/search_n_quant_SEARCHN/Identity:0'

- Scenario 5: The bottom layer of the network uses the BN small operator structure, whose input is 4-dimensional.
Messages similar to the following are displayed during graph modification:

2020-09-01 09:46:46,426 - WARNING - [AMCT]:[replace_bn_pass]: Replace BN at the end of the network! You need to replace the old output node by the new output node in inference process!
The name of the old output node is 'batch_normalization/batchnorm/add_1:0'
The name of the new output node is 'batch_normalization/batchnorm/bn_replace/batch_normalization/FusedBatchNormV3:0'

2020-09-01 09:46:46,439 - WARNING - [AMCT]:[conv_bn_fusion_pass]: Fused BN at the end of the network! You need to replace the old output node by the new output node in inference process!
The name of the old output node is 'batch_normalization/batchnorm/bn_replace/batch_normalization/FusedBatchNormV3:0'
The name of the new output node is 'bias_add:0'

2020-09-01 09:46:46,518 - WARNING - [AMCT]:[quantize_model]: Insert searchN operator at the end of the network! You need to replace the old output node by the new output node in inference process!
The name of the old output node is 'bias_add:0' //Original output node
The name of the new output node is 'search_n_quant/search_n_quant.SEARCHN/Identity:0' //New output node
Script Modification

If the `quantize_model` API is called to modify the original TensorFlow network graph, the output node at the bottom layer changes because a searchN layer is inserted at the end of the network. In this case, you need to modify the quantization script based on the log to replace the output node during network inference with the new node name as follows:

Quantization script before modification (The following script is only an example.)

```python
import tensorflow as tf
import amct_tensorflow as amct

def load_pb(model_name):
    with tf.gfile.GFile(model_name, "rb") as f:
        graph_def = tf.GraphDef()
        graph_def.ParseFromString(f.read())
        tf.import_graph_def(graph_def, name="")

def main():
    # Name of the network .pb file
    model_name = './pb_model/case_1_1.pb'
    # Name of the output node in network quantization inference
    infer_output_name = 'Add:0'
    # Name of the output node of the quantized model
    save_output_name = 'Add:0'

    # Load the .pb file of the network.
    load_pb(model_name)
    # Obtain the network graph.
    graph = tf.get_default_graph()

    # Generate a quantization configuration file.
    amct.create_quant_config(
        config_file='./configs/config.json',
        graph=graph)
    # Insert quantization operators.
    amct.quantize_model(
        graph=graph,
        config_file='./configs/config.json',
        record_file='./configs/record_scale_offset.txt')

    # Start network inference.
    with tf.Session() as sess:
        output_tensor = graph.get_tensor_by_name(infer_output_name)
        sess.run(tf.global_variables_initializer())
        sess.run(output_tensor)

    # Save the quantized .pb model file.
    amct.save_model(
        pb_model=model_name,
        outputs=[save_output_name[:-2]],
        record_file='./configs/record_scale_offset.txt',
        save_path='./pb_model/case_1_1')

if __name__ == '__main__':
    main()
```

The modified quantization script is as follows.

```python
import tensorflow as tf
import amct_tensorflow as amct

def load_pb(model_name):
    with tf.gfile.GFile(model_name, "rb") as f:
        graph_def = tf.GraphDef()
        graph_def.ParseFromString(f.read())
        tf.import_graph_def(graph_def, name="")

def main():
    # Name of the network .pb file
    model_name = './pb_model/case_1_1.pb'
    # Name of the output node in network quantization inference
    infer_output_name = 'Add:0'
    # Name of the output node of the quantized model
    save_output_name = 'Add:0'

    # Load the .pb file of the network.
    load_pb(model_name)
    # Obtain the network graph.
    graph = tf.get_default_graph()

    # Generate a quantization configuration file.
    amct.create_quant_config(
        config_file='./configs/config.json',
        graph=graph)
    # Insert quantization operators.
    amct.quantize_model(
        graph=graph,
        config_file='./configs/config.json',
        record_file='./configs/record_scale_offset.txt')

    # Start network inference.
    with tf.Session() as sess:
        output_tensor = graph.get_tensor_by_name(infer_output_name)
        sess.run(tf.global_variables_initializer())
        sess.run(output_tensor)

    # Save the quantized .pb model file.
    amct.save_model(
        pb_model=model_name,
        outputs=[save_output_name[:-2]],
        record_file='./configs/record_scale_offset.txt',
        save_path='./pb_model/case_1_1')

    if __name__ == '__main__':
        main()
```

def load_pb(model_name):
    with tf.gfile.GFile(model_name, "rb") as f:
        graph_def = tf.GraphDef()
        graph_def.ParseFromString(f.read())
        tf.import_graph_def(graph_def, name='')

def main():
    # Name of the network .pb file
    model_name = './pb_model/case_1_1.pb'
    # Name of the output node in quantization inference, which needs to be replaced with the new node name printed in the log.
    infer_output_name = 'search_n_quant/search_n_quant_SEARCHN/Identity:0'
    # Name of the output node of the quantized model
    save_output_name = 'Add:0'

    # Load the .pb file of the network.
    load_pb(model_name)
    # Obtain the network graph.
    graph = tf.get_default_graph()

    # Create a quantization configuration file.
    amct.create_quant_config(
        config_file='./configs/config.json',
        graph=graph)

    # Insert quantization operators.
    amct.quantize_model(
        graph=graph,
        config_file='./configs/config.json',
        record_file='./configs/record_scale_offset.txt')

    # Start network inference.
    with tf.Session() as sess:
        output_tensor = graph.get_tensor_by_name(infer_output_name)
        sess.run(tf.global_variables_initializer())
        sess.run(output_tensor)

    # Save the quantized .pb model file.
    amct.save_model(
        pb_model=model_name,
        outputs=[save_output_name[:-2]],
        record_file='./configs/record_scale_offset.txt',
        save_path='./pb_model/case_1_1')

if __name__ == '__main__':
    main()

5.11.7 How Do I Restore the Model Training Parameters After Quantization Operators Are Inserted?

The list of quantization variable operators (quant_add_ops) to be added has been passed to the 5.9.2.2 quantize_model API call. The variable values in the list cannot be found in the model training file. Therefore, an error indicating that the variables cannot be found is reported when the model training parameters are restored. In this case, you need to delete the variable values in the quant_add_ops list from the restoration list before restoring the model parameters.

1. Restoration of shadow variables
   # 1. Obtain the dictionary variables_dict of each {key:value} variable.
   variables_ema = tf.train.ExponentialMovingAverage(moving_average_decay)
   variables_dict = variables_ema.variables_to_restore()
2. Define the variables to be restored. \textbf{\{key:value\}} corresponds to the dictionary \textbf{params_need_load}.
\begin{verbatim}
params_need_load = dict()
\end{verbatim}

3. Find the variables to be restored from \textbf{variables_dict} based on \textbf{quant_add_ops}.
\begin{verbatim}
for key, value in variables_dict.items():
    if value not in quant_add_ops:
        params_need_load[key] = value
\end{verbatim}

4. Restore variables.
\begin{verbatim}
loader = tf.train.Saver(params_need_load)
loader.restore(sess, FLAGS.checkpoint)
\end{verbatim}

## 5.11.8 Python 3.7.5 Installation on Ubuntu

### Step 1
Check that the Python 3.7.5 development environment is available.

Run the \textbf{python3.7.5 --version}, \textbf{python3.7 --version}, \textbf{pip3.7.5 --version}, and \textbf{pip3.7 --version} commands to check whether the environment is available. If the following information is displayed, the environment is available. Otherwise, go to the next step.

\begin{verbatim}
Python 3.7.5
pip 19.2.3 from /usr/local/python3.7.5/lib/python3.7/site-packages/pip (python 3.7)
\end{verbatim}

### Step 2
Install the Python 3.7.5 dependencies.

\begin{verbatim}
sudo apt-get install -y make zlib1g zlib1g-dev build-essential libbz2-dev libsqlite3-dev libssl-dev libxml2-dev libffi-dev openssl python3-tk
\end{verbatim}

\textbf{libsqlite3-dev} must be installed before the Python installation. If the Python 3.7.5 environment has been installed in the user’s OS before the libsqlite3-dev installation, you need to rebuild the Python environment. If python3-tk fails to be installed, see \textbf{5.10.1 An Error Message Is Displayed During python3-tk Installation}.

### Step 3
Install Python 3.7.5.

1. Run the \textbf{wget} command to download the source package of Python 3.7.5 to any directory on the server where AMCT is located:
\begin{verbatim}
wget https://www.python.org/ftp/python/3.7.5/Python-3.7.5.tgz
\end{verbatim}

2. Go to the download directory and decompress the source package:
\begin{verbatim}
tar -zxvf Python-3.7.5.tgz
\end{verbatim}

3. Go to the new folder and run the following configuration, build, and installation commands:
\begin{verbatim}
cd Python-3.7.5
./configure --prefix=/usr/local/python3.7.5 --enable-loadable-sqlite-extensions --enable-shared
\end{verbatim}
make
sudo make install

--prefix specifies the Python installation path. You can modify it as required.
--enable-shared is used to build the libpython3.7m.so.1.0 dynamic library.
--enable-loadable-sqlite-extensions is used to load the sqlite-devel dependency.

This document uses --prefix=/usr/local/python3.7.5 as an example. After the configuration, build, and installation commands are executed, the installation package is output to the /usr/local/python3.7.5 directory, and the libpython3.7m.so.1.0 dynamic library is output to the /usr/local/python3.7.5/lib/libpython3.7m.so.1.0 directory.

4. Set the soft links:
sudo ln -s /usr/local/python3.7.5/bin/python3 /usr/local/python3.7.5/bin/python3.7.5
sudo ln -s /usr/local/python3.7.5/bin/pip3 /usr/local/python3.7.5/bin/pip3.7.5

5. Set the Python 3.7.5 environment variables.
   a. If Python is installed by the root user:

   AMCT is installed by the root user. Run the following commands in the current terminal window to set environment variables:

   # Set the Python 3.7.5 library path.
   export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
   # If multiple Python 3 versions exist in the user environment, specify Python 3.7.5.
   export PATH=/usr/local/python3.7.5/bin:$PATH

   NOTICE

   If the running user is the root user, it is not advised to modify the .bashrc file. Otherwise, the Python tools provided by other systems may be unavailable. If you want to use the default tool, open another terminal window.

   b. If Python is installed by a non-root user:

   AMCT is also installed by the non-root user. Run the vi ~/.bashrc command in any directory to open the .bashrc file, and append the following lines to the file:

   # Set the Python 3.7.5 library path.
   export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
   # If multiple Python 3 versions exist in the user environment, specify Python 3.7.5.
   export PATH=/usr/local/python3.7.5/bin:$PATH

   Run the .wq! command to save the file and exit. Run the source ~/.bashrc command for the modification to take effect immediately.

   Step 4 After the installation is complete, run the following commands to check the installed version. If the required version information is displayed, the installation is successful.

   python3.7.5 --version
   pip3.7.5 --version
   python3.7 --version
   pip3.7 --version

   -----End
6 AMCT Instructions (ONNX)

6.1 Introduction

6.1.1 Overview

This document describes how to quantize an ONNX model using Ascend Model Compression Toolkit (AMCT). In the quantization process, the precision of model weights and activations is reduced to make the model more compact, improving the compute efficiency while lowering the transfer latency.

AMCT is an ONNX+ONNX Runtime-based Python toolkit that implements Conv+BN fusion (Conv+BN fusion is performed on the "Conv+BatchNormalization" composite in the model before AMCT quantization) as well as 8-bit quantization of activations and weights in neural networks. This toolkit decouples model quantization from model conversion. It implements independent quantization of quantization-capable operators in a model, and outputs an .onnx model file. The obtained accuracy simulation model can run on CPU or GPU to complete accuracy simulation. The obtained deployable model can run on the Ascend AI Processor with improved inference performance. This tool has the following advantages:

- Lightweight: You only need to install the tool.
- Easy-to-use APIs: You can complete quantization using APIs based on the ONNX Runtime inference script.
6.1.2 Features

6.1.2.1 Post-training Quantization

6.1.2.1.1 Terminology

There are two forms of quantization: post-training quantization and quantization aware training. In this document, quantization is performed after training, which is referred to as post-training quantization. The weights of a trained model are quantized from float32 to int8 and the activations are calibrated and quantized using a small calibration dataset. For details about the quantization workflow, see 6.3 Post-training Quantization. The quantization form is classified into weight quantization and activation quantization according to the quantization object.

- **Calibration dataset**
  During the calibration process, the algorithm uses each piece of data in the calibration dataset as input, accumulates the input data of each layer (or operator) that needs to be quantized, and uses the accumulated data as input data of the quantization algorithm to determine the quantization factors. Because the quantization factors and the accuracy of the quantized model are closely related to the selection of the calibration dataset, you are advised to calibrate the model using a subset of images from the validation dataset.

- **Activation quantization**
Activation quantization is to collect statistics on the input data of each layer (or operator) to be quantized, to find the optimal pair of scale and offset per layer (or operator). For details, see 6.8.2 Quantization Factor Record File. Activations are the intermediate results of model inference computation. The value ranges are input-specific. Therefore, a group of reference inputs (a calibration dataset) needs to be used as incentives to record the input data of each layer (or operator) to be quantized for searching for quantization factors (scale and offset). During data calibration, extra memory (video memory/RAM) is needed to store the input data used to determine the quantization factors. Therefore, the video memory/RAM usage is higher than that required for performing inference only. The size of the extra memory is positively correlated with $\text{batch}_\text{size} \times \text{batch}_\text{num}$ during calibration.

- **Weight quantization**
  After model training, the weights and the value ranges are determined. Therefore, quantization may be directly performed based on the value range of each weight.

The layers that support quantization are listed as follows.

**Table 6-1** Layers that support quantization and restrictions

<table>
<thead>
<tr>
<th>Layer</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv</td>
<td>-</td>
</tr>
<tr>
<td>Gemm</td>
<td>transpose_a = false, Alpha = Beta = 1.0</td>
</tr>
<tr>
<td>MatMul</td>
<td>Quantization is supported only when the weight has rank 2.</td>
</tr>
</tbody>
</table>

**6.1.2.1.2 Principles**

Figure 6-2 shows the AMCT principles. The blue parts are implemented by the user, and the gray part is implemented by the call to AMCT's 6.6.4 `convert_qat_model` API.

1. Construct an original ONNX model and then call the 6.6.1 `create_quant_config` API to generate a quantization configuration file.
2. Call the 6.6.2 `quantize_model` API to optimize the original ONNX model based on the ONNX model and quantization configuration file, including Conv +BN fusion. Next, quantize the weights and inserts activation quantization nodes to the model. Then, perform inference with the quantized model in the ONNX Runtime environment based on calibration dataset.
3. Call the 6.6.3 `save_model` API to save the quantized model to a model for accuracy simulation in the ONNX Runtime environment and a model deployable on the Ascend AI Processor.

For details about APIs, see 6.6 APIs.
6.1.2.2 QAT Model to Ascend Model Conversion

Overview

An already-quantized original ONNX model is referred to as a QAT model. Prior to generating an offline model adapted to the Ascend AI Processor with ATC, you need to use AMCT's 6.6.4 convert_qat_model API to convert the QAT model into an Ascend representation, which means that ATC does not support direct conversion from a QAT model to an offline model. Note the following restrictions on QAT model to Ascend model conversion:

- The source QAT model must have FakeQuant layers (including QuantizeLinear and DequantizeLinear). Per-channel quantization takes effect on weights only. A QuantizeLinear-DequantizeLinear layer pair must have the same quantization factor.
- Only the Conv, Gemm, and MatMul layers can match fake_quant nodes, which means only these layers are converted. For details about the layer restrictions, see Table 6-1.

Principles

Figure 6-3 shows the conversion principles. In the figure, the blue part is implemented by the user, and the gray part is implemented by the call to AMCT's 6.6.4 convert_qat_model API. The user needs to import the dependence library to the ONNX QAT model inference code and inserts the corresponding API call to implement the conversion.

Figure 6-3 QAT model to Ascend model conversion

6.1.3 Tool Workflow

Figure 6-4 shows the tool workflow.
Figure 6-4 Tool workflow

Table 6-2 Major actions in the tool workflow

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package preparation</td>
<td>Obtain the tool package by referring to 6.2.1 Package Preparation.</td>
</tr>
<tr>
<td>Pre-installation actions</td>
<td>Before AMCT installation, create an installation user, check the system environment, install dependencies, and upload the AMCT package. For details, see 6.2.2 Pre-installation Actions.</td>
</tr>
<tr>
<td>Installation</td>
<td>Install the ONNX AMCT by referring to 6.2.3 Installation.</td>
</tr>
<tr>
<td>Post-installation actions</td>
<td>AMCT depends on a custom operator package (OPP) based on the ONNX Runtime, and the building of custom operators depends on the header files provided by the ONNX Runtime. Therefore, you need to download the related header files, build and install the custom OPP. For details, see 6.2.4 Post-installation Actions. To set the quantization logging level, you need to set related environment variables.</td>
</tr>
<tr>
<td>Action</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>(Optional) Script creation with AMCT API calls</td>
<td>If you need to quantize your network model instead of the sample model provided in this instruction, you need to modify the script for adaptation before quantization.</td>
</tr>
<tr>
<td>Quantization</td>
<td>See 6.3 Post-training Quantization.</td>
</tr>
<tr>
<td>(Optional) Model conversion using ATC</td>
<td>You can convert the quantized deployable model to an offline model supported by the Ascend AI Processor by using ATC, and then perform subsequent inference.</td>
</tr>
</tbody>
</table>

## 6.2 AMCT Installation

### 6.2.1 Package Preparation

Currently, AMCT runs only on Ubuntu 18.04 (x86_64) servers. Before installation, click [here](#) to obtain the AMCT package `Ascend-cann-amct_{software version}_ubuntu18.04-x86_64.tar.gz`.

Before installation, obtain the AMCT package. AMCT runs on Ubuntu 18.04 (x86_64) or EulerOS (AArch64) servers. Select a required software package.

- Ubuntu 18.04 (x86_64) server: `Ascend-amct_{software version}_ubuntu18.04-x86_64.tar.gz`
- EulerOS (AArch64) server: `Ascend-amct_{software version}_euleros2.9.aarch64.tar.gz`

`{software version}` indicates the version number.

### 6.2.2 Pre-installation Actions

#### 6.2.2.1 Ubuntu (x86)

**Preparing the AMCT User**

Any user (root or non-root) is allowed to install AMCT. This section uses a non-root user as an example.

- To install AMCT as the root user, skip this section.
- To install AMCT as an existing non-root user, ensure that the user has the read, write, and execute permissions on the $HOME directory.
- To install AMCT as a new non-root user, perform the following steps as the root user. The following uses this scenario as an example.
  a. Create an AMCT installation user and set the $HOME directory for the user:
useradd -d /home/username -m username

b. Set the user password:

pwd /username

NOTE

username indicates the name of the AMCT installation user. The umask value of the user is at least 0027.

- You can view the umask value by running the umask command.
- You can change the umask value by running the umask NewValue command.

(Optional) Setting the Permission of the AMCT Installation User

Skip this part if you install AMCT as the root user.

Before installing AMCT, you need to download the dependencies, which requires the sudo apt-get permission. Run the following commands as the root user:

1. Open the /etc/sudoers file:

   chmod u+w /etc/sudoers
   vi /etc/sudoers

2. Add the following content under # User privilege specification in the file:

   username ALL=(ALL:ALL)   NOPASSWD:SETENV:/usr/bin/apt-get,/usr/bin/pip, /bin/tar, /bin/mkdir, /bin/sh, /bin/bash, /usr/bin/locate, /usr/bin/make, /usr/bin/pip3, /usr/bin/pip3.7, /usr/bin/pip3.7.5, /bin/ln

   Replace username with the name of the non-root user who executes the installation script.

   NOTE

   Check if the last line in the /etc/sudoers file is #includedir /etc/sudoers.d. If no, add it manually.

3. Run the :wq! command to save the file.

4. Remove the write permission on the /etc/sudoers file:

   chmod u-w /etc/sudoers

Setting Up Environment

NOTE

This tool supports Python 3.7.0 to Python 3.7.9. This document uses Python 3.7.5 as an example. The environment variables and installation commands vary according to the actual Python version.

Currently, AMCT runs on Ubuntu 18.04 (x86_64). The following table details the architecture mapping.

<table>
<thead>
<tr>
<th>Item</th>
<th>Version</th>
<th>How to Obtain</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Ubuntu 18.04 (64-bit)</td>
<td>Click here to download an Ubuntu release. The ubuntu-18.04-server-amd64.iso server install image is recommended.</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6-3 Ubuntu (x86_64) architecture mapping
<table>
<thead>
<tr>
<th>Item</th>
<th>Version</th>
<th>How to Obtain</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>3.7.5</td>
<td>See 6.8.3 Python 3.7.5 Installation on Ubuntu</td>
<td>Before running the <code>apt-get</code> command to install the dependencies, ensure that the server can access the Internet.</td>
</tr>
<tr>
<td>ONNX and opset</td>
<td>ONNX 1.7.0, opset v11</td>
<td>See Installing Dependencies.</td>
<td>Only operators in ai.onnx opset v11 are supported.</td>
</tr>
<tr>
<td>ONNX Runtime</td>
<td>1.5.2</td>
<td>ONNX Runtime framework. For details, see Installing Dependencies. The framework provides the CPU version and GPU version. Choose one as needed. Note that the GPU version depends on CUDA 10.2 and cuDNN 8.0.3.</td>
<td>-</td>
</tr>
<tr>
<td>OpenCV-Python</td>
<td>4.2.0.32</td>
<td>See Installing Dependencies.</td>
<td>-</td>
</tr>
<tr>
<td>NumPy</td>
<td>1.16.0+</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Protobuf</td>
<td>3.11.0+</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

**Checking Sources**

During dependency installation, you need to make sure that the server of AMCT has Internet access. Run the following command as the root user to check source validity:

```
apt-get update
```

If an error is reported during the command execution, check whether the network connection is normal or replace the source in the `/etc/apt/sources.list` file with a valid one.
Installing Dependencies

Use the AMCT installation user to install software. If the installation user is a non-root user, run the **su - username** command to switch to the non-root user and run the following commands.

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Version</th>
<th>Installation Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>3.7.5</td>
<td>See 6.8.3 Python 3.7.5 Installation on Ubuntu.</td>
</tr>
<tr>
<td>ONNX</td>
<td>1.7.0</td>
<td>pip3.7.5 install onnx==1.7.0 --user</td>
</tr>
</tbody>
</table>
| ONNX Runtime | 1.5.2 | ● # Install the ONNX Runtime CPU version. pip3.7.5 install onnxruntime==1.5.2 --user  
|            |         | ● # Install the ONNX Runtime GPU version. Before installing the GPU version, install CUDA 10.2 and cuDNN 8.0.3. pip3.7.5 install onnxruntime-gpu==1.5.2 --user |
| OpenCV-Python | 4.2.0.32 | pip3.7.5 install opencv-python==4.2.0.32 --user |
| NumPy      | 1.16.0+ | pip3.7.5 install numpy==1.16.0 --user |
| Protobuf   | 3.11.0+ | pip3.7.5 install protobuf==3.11.0 --user |

### 6.2.3 Installation

**Step 1** Install AMCT.

In the directory where the AMCT package is located, run the following command:

```
pip3.7.5 install amct_onnx-{version}-py3-none-linux_{arch}.whl --user
```

Replace `{version}` with the actual AMCT version number, and `{arch}` with the actual architecture of the installation server. If AMCT installation is performed by the root user and the --target option is included, ensure that the path specified by --target is the path of the current user.

**Step 2** Check the installation. If a message similar to the following is displayed, the installation is successful:

```
Successfully installed amct-onnx-{version}
```

Find the installed AMCT in the python3.7.5 directory (for example, SHOME/.local/lib/python3.7.5/site-packages).

```
drwxr-xr-x  5 amct  amct  4096 Mar 17 11:50 amct_onnx/
drwxr-xr-x  2 amct  amct  4096 Mar 17 11:50 amct_onnx-{version}.dist-info/
```

`amct_onnx` indicates the AMCT installation path.

---End
6.2.4 Post-installation Actions

6.2.4.1 Building and Installing Custom OPP

AMCT depends on a custom operator package (OPP) based on the ONNX Runtime, while building a custom OPP depends on the ONNX Runtime header files. You need to download the header files, and then build and install a custom OPP as follows.

1. Decompress the custom OPP package.

```bash
tar -zxvf amct_onnx_op.tar.gz
```

The directory structure after decompression is as follows:

| amct_onnx_op          # Custom OPP root directory |
|-----------------------# Directory of the header files for custom OPP building. |
| ├── inc               # Directory of the header files for custom OPP building. |
| │   └── __init__.py    # Script for downloading the header files required for building an ONNX Runtime-based custom OPP. To use this script, ensure that the environment has Internet access. |
| │   └── download_inc_files.py # AMCT quantization algorithm declaration |
| ├── quant.h           # AMCT calibration auxiliary function declaration |
| ├── util.h            # Directory of source files for custom operator implementation. For details, see the ONNX Runtime API tutorial. |
| ├── src               # Directory of source files for custom operator implementation. For details, see the ONNX Runtime API tutorial. |
| │   ├── ifmr_op_library.cpp # Source file of the IFMR activation quantization operator functions |
| │   └── ifmr_op_library.h  # IFMR activation quantization operator declaration |
| └── setup.py          # Build script that builds custom operators and copies the generated dynamic libraries to the AMCT package |

**NOTE**

The building of ONNX Runtime-based custom operators depends on the header files provided by ONNX Runtime. Download them from GitHub. If the server where AMCT is located has Internet access and can visit GitHub, go to 2. Otherwise, manually download the following files and upload them to the `amct_onnx_op/inc` directory on the AMCT server:

- [https://raw.githubusercontent.com/microsoft/onnxruntime/v1.5.2/include/onnxruntime/core/session/onnxruntime_cxx_api.h](https://raw.githubusercontent.com/microsoft/onnxruntime/v1.5.2/include/onnxruntime/core/session/onnxruntime_cxx_api.h)
- [https://raw.githubusercontent.com/microsoft/onnxruntime/v1.5.2/include/onnxruntime/core/session/onnxruntime_cxx_inline.h](https://raw.githubusercontent.com/microsoft/onnxruntime/v1.5.2/include/onnxruntime/core/session/onnxruntime_cxx_inline.h)
- [https://raw.githubusercontent.com/microsoft/onnxruntime/v1.5.2/include/onnxruntime/core/session/onnxruntime_c_api.h](https://raw.githubusercontent.com/microsoft/onnxruntime/v1.5.2/include/onnxruntime/core/session/onnxruntime_c_api.h)
- [https://raw.githubusercontent.com/microsoft/onnxruntime/v1.5.2/include/onnxruntime/core/session/onnxruntime_session_options_config_keys.h](https://raw.githubusercontent.com/microsoft/onnxruntime/v1.5.2/include/onnxruntime/core/session/onnxruntime_session_options_config_keys.h)

2. Go to the `amct_onnx_op` directory and build and install a custom OPP.

```bash
cd amct_onnx_op && python3.7.5 setup.py build
```

If messages similar to the following are displayed, the custom OPP has been built and installed successfully for AMCT:

```
[INFO] Build amct_onnx_op success!
[INFO] Install amct_onnx_op success!
```

6.2.4.2 Setting Environment Variables

You can set the AMCT quantization log level using environment variables. Logs include the logs printed to the screen and the logs saved in the `amct_log/amct_onnx.log` file. The environment variables are optional. If they are not set, the default log level INFO is used.

- **Variables**
The log level is set by the following variables:

- **AMCT_LOG_FILE_LEVEL**: specifies the level of messages in the `amct_onnx.log` file and the level of messages in the log file generated of the corresponding quantization layer when the model for accuracy simulation is generated.
- **AMCT_LOG_LEVEL**: specifies the level of log messages printed to the screen.

*Table 6-5* lists the valid values and their meanings.

### Table 6-5 Variable description

<table>
<thead>
<tr>
<th>Logging Level</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG</td>
<td>Outputs DEBUG, INFO, WARNING, and ERROR logs.</td>
<td>Detailed process messages, including the quantization layer and corresponding processing phase (fusion, parameter quantization, or activation quantization)</td>
</tr>
<tr>
<td>INFO</td>
<td>Outputs INFO, WARNING, and ERROR logs. The default value is INFO.</td>
<td>Brief quantization processing messages, including the quantization phase.</td>
</tr>
<tr>
<td>WARNING</td>
<td>Outputs WARNING and ERROR logs.</td>
<td>Warning messages during quantization.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Outputs ERROR logs.</td>
<td>Error messages during quantization.</td>
</tr>
</tbody>
</table>

The logging level is case insensitive. That is, *Info*, *info*, and *INFO* are all valid values.

- **Examples**

The following commands are only examples. You can set the level as required.

- Set the quantization log level of `amct_onnx.log` to *INFO*.
  
  ```bash
  export AMCT_LOG_FILE_LEVEL=INFO
  ```

- Set the level of the information displayed on the screen to *INFO*.
  
  ```bash
  export AMCT_LOG_LEVEL=INFO
  ```

### 6.3 Post-training Quantization
6.3.1 Prerequisites

Model

Upload the ONNX model to be quantized to any directory on the Linux server as the AMCT installation user. The following uses the ResNet-101 model available in the sample package as an example.

If you choose to use your own model, you are advised to perform inference in the ONNX Runtime environment in advance to test if it can run properly in the environment with expected accuracy.

Image Dataset

After the model is quantized using the AMCT, perform inference with the model to test the model accuracy. Use the dataset that matches the model.

Upload the dataset matching the model to any directory on the Linux server as the AMCT installation user. The following uses the images dataset (which also serves as the calibration dataset) preset in the sample as an example.

Calibration Dataset

The calibration dataset is used to generate the quantization factors to guarantee the accuracy.

The process of calculating the quantization factors is referred to as calibration. Perform inference using the quantized network with one or more batches of a subset of images from the validation dataset to complete calibration.

Upload the calibration dataset file to any directory on the Linux server as the AMCT installation user.

6.3.2 Quantization Example

The following uses the image classification network's quantization script resnet101_sample.py, original ONNX model, image dataset (images), and calibration dataset (images) to illustrate how to execute the quantization script.

1. Obtain the quantization script.

   In the directory of amct_onnx_sample.tar.gz, extract the quantization script from the package:
   ```
tar -zxvf amct_onnx_sample.tar.gz
cd sample
   ```

   Find the following extracted files and directories:
   - resnet101/resnet101_sample.py: quantization sample script for the image classification network.
   - resnet101/images/: image dataset (also the calibration dataset) for the image classification network, containing 160 images.

2. Run the quantization script.
Run the following command in the `sample/resnet101` directory to quantize the ResNet-101 network:

```
python3.7.5 resnet101_sample.py
```

**NOTE**

The preceding quantization command applies to single-GPU or non-GPU environments. If there are multiple GPUs, run the following command to specify the target GPU:

```
CUDA_VISIBLE_DEVICES=0 python3.7.5 resnet101_sample.py
```

If messages similar to the following are displayed, the quantization is successful:

```
******top1:0.76875 ******top5:0.925    //Top 1 and top 5 inference accuracy results of the quantized fake_quant model in the ONNX Runtime environment.
[INFO] ResNet101 before quantize top1: 0.775 top5: 0.91875  //Inference result of the original model. It is an example only.
[INFO] ResNet101 after quantize top1: 0.76875 top5: 0.925    //Inference result of the quantized model. It is an example only.
```

3. View the quantization result.

After the quantization is complete, find the following files generated in the directory of the quantization script:

- `tmp`: temporary directory, containing:
  - `config.json`: configuration file that describes how to quantize each layer in the model. If a quantization configuration file already exists in the directory of the quantization script, the existing quantization configuration file is overwritten by a new one with the same name in a call to 6.6.1 `create_quant_config`. If not that case, a new quantization configuration file is created.
  
  If the accuracy of model inference drops significantly after quantization, you can modify the `config.json` file by referring to 6.3.3 `Quantization Configuration`.
  
  - `modified_model.onnx`: calibrated model file returned by the `quantize_model` API call.
  - `scale_offset_record.txt`: file that records quantization factors. For details about the prototype definition of the file, see 6.8.2 `Quantization Factor Record File`.
  
  - `results/calibration_results`: quantization result directory, containing:
    
    - `resnet101_deploy_model.onnx`: quantized model file to be deployed on the Ascend AI Processor.
    
    - `resnet101_fake_quant_model.onnx`: quantized model file that can be used for accuracy simulation in the ONNX Runtime environment.

4. (Optional) Convert the quantized model into an offline model adapted to the Ascend AI Processor by referring to:

`ATC Instructions`
6.3.3 Quantization Configuration

This section describes the quantization configuration file of image classification networks.

6.3.3.1 Overview

If the inference accuracy of the config.json quantization configuration file generated by the 6.6.1 create_quant_config call does not meet the requirements, you need to tune the config.json file until the accuracy is as expected. The following is an example of the file content.

```
{
    "version":1,
    "batch_num":2,
    "activation_offset":true,
    "do_fusion":true,
    "skip_fusion_layers":[],
    "layer_name1":{
        "quant_enable":true,
        "activation_quant_params":{
            "max_percentile":0.999999,
            "min_percentile":0.999999,
            "search_range":{
                0.7,
                1.3
            },
            "search_step":0.01
        },
        "weight_quant_params":{
            "channel_wise":true
        }
    },
    "layer_name2":{
        "quant_enable":true,
        "activation_quant_params":{
            "max_percentile":0.999999,
            "min_percentile":0.999999,
            "search_range":{
                0.7,
                1.3
            },
            "search_step":0.01
        },
        "weight_quant_params":{
            "channel_wise":false
        }
    }
}
```

6.3.3.2 Configuration File Options

The following tables describe the parameters in the configuration file.

Table 6-6 version

<table>
<thead>
<tr>
<th>Function</th>
<th>Version number of the quantization configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>1</td>
</tr>
</tbody>
</table>

CANN
Development Auxiliary Tool Guide (Training)  6 AMCT Instructions (ONNX)
Currently, only version 1 is available.

**Recommended Value**: 1

**Required/Optional**: Optional

<table>
<thead>
<tr>
<th>Description</th>
<th>Batch count for quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>Batch count for quantization</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>int</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>Greater than 0</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If this option is not set, the default value 1 is used. It is recommended that the number of images in the calibration dataset be less than or equal to 50. The value of batch_num is calculated based on the value of batch_size. batch_num x batch_size equals the number of images in the calibration dataset used for quantization. batch_size indicates the number of images per batch.</td>
</tr>
</tbody>
</table>

**Recommended Value**: 1

**Required/Optional**: Optional

**Table 6-8 activation_offset**

<table>
<thead>
<tr>
<th>Function</th>
<th>Symmetric quantization or asymmetric quantization select for activation quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>true or false</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If it is set to true, asymmetric quantization is used. If it is set to false, symmetric quantization is used.</td>
</tr>
</tbody>
</table>

**Recommended Value**: true

**Required/Optional**: Optional
### Table 6-9 do_fusion

<table>
<thead>
<tr>
<th>Function</th>
<th>Fusion switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td>If it is set to true, fusion is enabled. If it is set to false, fusion is disabled. Currently, only Conv+BN fusion is supported.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>true</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 6-10 skip_fusion_layers

<table>
<thead>
<tr>
<th>Function</th>
<th>Layer skip in fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value Range</td>
<td>Layers that support fusion. Currently, only Conv+BN fusion is supported.</td>
</tr>
<tr>
<td>Description</td>
<td>Layers to skip in fusion</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>-</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 6-11 layer_config

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization configuration of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>Includes the following parameters:</td>
</tr>
<tr>
<td></td>
<td>• quant_enable</td>
</tr>
<tr>
<td></td>
<td>• activation_quant_params</td>
</tr>
<tr>
<td></td>
<td>• weight_quant_params</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>None</td>
</tr>
</tbody>
</table>
### Table 6-12 quant_enable

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Optional</th>
</tr>
</thead>
</table>

**Function**: Quantization enable by layer

**Type**: `bool`

**Value Range**: `true` or `false`

**Description**: If it is set to `true`, the layer is to be quantized. If it is set to `false`, otherwise.

**Recommended Value**: `true`

### Table 6-13 activation_quant_params

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Optional</th>
</tr>
</thead>
</table>

**Function**: Activation quantization parameters of a layer

**Type**: `object`

**Value Range**: None

**Description**: Includes the following parameters:
- `max_percentile`
- `min_percentile`
- `search_range`
- `search_step`

**Recommended Value**: None

### Table 6-14 weight_quant_params

<table>
<thead>
<tr>
<th>Required/Optional</th>
<th>Optional</th>
</tr>
</thead>
</table>

**Function**: Weight quantization parameters of a layer

**Type**: `object`

**Value Range**: None
<table>
<thead>
<tr>
<th>Description</th>
<th>Includes the following parameter: channel_wise</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recommended Value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 6-15 max_percentile**

<table>
<thead>
<tr>
<th>Function</th>
<th>Upper search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>float</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>(0.5, 1]</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Indicates the maximum number to be considered as the search result among a group of numbers in descending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the maximum, that is, the largest number. A larger value indicates that the upper clip limit is closer to the maximum value of the data to be quantized.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>0.999999</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 6-16 min_percentile**

<table>
<thead>
<tr>
<th>Function</th>
<th>Lower search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>float</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>(0.5, 1]</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Indicates the minimum number to be considered as the search result among a group of numbers in ascending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the minimum, that is, the smallest number. A larger value indicates that the lower clip limit is closer to the minimum value of the data to be quantized.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>0.999999</td>
</tr>
</tbody>
</table>
Table 6-17 search_range

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search range: ([\text{search_range_start}, \text{search_range_end}])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>A list of two floats</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0 &lt; \text{search_range_start} &lt; \text{search_range_end})</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the quantization factor search range:</td>
</tr>
<tr>
<td></td>
<td>● (\text{search_range_start}): search start.</td>
</tr>
<tr>
<td></td>
<td>● (\text{search_range_end}) search end.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>[0.7, 1.3]</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 6-18 search_step

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>((0, \text{(search_range_end} - \text{search_range_start})])</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the quantization factor search step. A smaller value indicates a smaller step.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>0.01</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 6-19 channel_wise

<table>
<thead>
<tr>
<th>Function</th>
<th>Whether to use different quantization factors for each channel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
</tbody>
</table>
6.3.3.3 Configuration Tuning

If the inference accuracy of the model quantized based on the default `config.json` configuration drops significantly, perform the following steps to tune the quantization configuration file:

**Step 1** Execute the quantization script in the `amct_onnx_sample.tar.gz` package to perform quantization based on the default configuration generated by the `create_quant_config` API.

**Step 2** If the inference accuracy with the model quantized in **Step 1** is as expected, configuration tuning ends. Otherwise, go to **Step 3**.

**Step 3** Tune `batch_num` in the quantization configuration file.

`batch_num` controls the batch count for quantization. Tune it based on the batch size and the number of images required for quantization. Generally, a larger quantity of data samples used in a quantization process indicates a smaller accuracy loss after quantization. However, excessive data does not necessarily improve accuracy, but certainly consumes more memory and reduces the quantization speed, hence resulting in insufficient memory, video RAM, and thread resources. Therefore, it is recommended that the product of `batch_num` and `batch_size` be 16 or 32.

**Step 4** If the inference accuracy with the model quantized in **Step 3** is as expected, configuration tuning ends. Otherwise, go to **Step 5**.

**Step 5** Tune `quant_enable` in the quantization configuration file.

`quant_enable` specifies whether to quantize a layer. If set to `true`, the layer is to be quantized. If set to `false`, otherwise. If the configuration of a layer is not present, the quantization of the layer is skipped. Generally, specifying fewer layers to quantize improves quantization accuracy. When the network accuracy is not as expected, locate the quantization-sensitive layers (whose error increases significantly after quantization, such as the top layer, bottom layer, depthwise convolutional layer, and layers with few parameters) in the network, and disable quantization on these layers as needed.

**Step 6** If the inference accuracy with the model quantized in **Step 5** is as expected, configuration tuning ends. Otherwise, go to **Step 7**.

**Step 7** Tune the values of `activation_quant_params` and `weight_quant_params` in the quantization configuration file.
Data is clipped to the range \([left, right]\) specified by the \texttt{activation_quant_params} parameters. Generally, values distributed near a boundary are sparse, and clip may be performed on all the values, to improve the accuracy. A larger value of \texttt{min_percentile (max_percentile)} indicates that \texttt{left (right)} is closer to the minimum value (maximum value) of the to-be-quantized data. \texttt{search_range} and \texttt{search_step} affect the range of \([left, right]\). Generally, a larger value of \texttt{search_range} and a smaller value of \texttt{search_step} may achieve higher quantization accuracy, but the quantization takes more time.

\textbullet \hspace{1em} \texttt{channel_wise} in \texttt{weight_quant_params} determines whether to use a different quantization factor for each channel during weight quantization. If set to \texttt{true}, channels are separately quantized using different quantization factors. If set to \texttt{false}, all channels are quantized altogether using the same quantization factors. Generally, the inference accuracy is higher if the channels are separately quantized. However, the MatMul layer is channel-irrelevant. Therefore, this parameter does not take effect on the layer.

**Step 8** If the inference accuracy with the model quantized in **Step 7** is as expected, configuration tuning ends. Otherwise, it indicates that quantization has severe adverse impact on the inference accuracy. In this case, remove the quantization configuration.

----End
6.4 AMCT Update

The latest AMCT release allows you to access to the latest features. Before updating AMCT, uninstall the existing installation by referring to 6.5 AMCT
Uninstallation, and then install the latest version by referring to 6.2 AMCT Installation.

6.5 AMCT Uninstallation

You can uninstall AMCT as follows:

1. Run the following command in any directory on the Linux server as the AMCT installation user:
   ```bash
   pip3.7.5 uninstall amct_onnx
   ```
2. When the following information is displayed, enter y:
   ```console
   Uninstalling amct-onnx-{version}:
   Would remove:
   ...
   ...
   Proceed (y/n)? y
   ```
   If a message similar to the following is displayed, the uninstallation is successful:
   ```console
   Successfully uninstalled amct-onnx-{version}
   ```

6.6 APIs

6.6.1 create_quant_config

Description

Finds all quantization-capable layers in a graph, creates a quantization configuration file, and writes the quantization configuration information of the quantization-capable layers to the configuration file.

Prototype

```python
create_quant_config(config_file, model_file, skip_layers=None, batch_num=1, activation_offset=True, config_defination=None, updated_model=None)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the quantization configuration file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The existing file (if available) in the directory will be overwritten upon</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>this API call.</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>model_file</td>
<td>Input</td>
<td>ONNX model file to quantize. The model must be generated based on ONNX opset 11 and is inference-capable on ONNX Runtime 1.5.2.</td>
<td>A string.</td>
</tr>
</tbody>
</table>
| skip_layers   | Input        | Quantization-capable layers to skip.                                                                                                       | Default: **None**  
A list of strings.  
Restriction: If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect. |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
</table>
| batch_num      | Input        | Number of batches taken to generate the quantization factors.                | An int.                                                                                                                                          Value range: any integer larger than 0. Default: 1                                                                                      Restrictions:  
  - batch_num cannot be too large. The product of batch_num and batch_size equals to the number of images used during quantization. Too many images consume too much memory.  
  - If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect. |
| activation_offset | Input        | Whether to quantize activations with offset.                                 | Default: True                                                                                 A bool.                                                                                                                                   Restriction: If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect. |
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
</table>
| config_defination | Input | Whether to create a simplified quantization configuration file `quant.cfg` from the `calibration_config_onnx.proto` file in `/amct_pytorch/proto/calibration_config_onnx.proto` under the AMCT installation path. For details about the parameters in the `calibration_config_onnx.proto` template and the generated simplified quantization configuration file `quant.cfg`, see [6.8.1 Simplified Post-training Quantization Configuration File](#). | Default: **None**  
A string.  
Restriction: If it is set to **None**, a configuration file is generated based on the residual arguments (`skip_layers`, `batch_num`, and `activation_offset`). Otherwise, a configuration file in JSON format is generated based on this argument. |
| updated_model | Input | If this parameter is specified, the node names in the model are updated. If a node does not have a name, a unique name is generated in the `{op_type}_{index}` format. For nodes with duplicate names, digit suffixes are added to ensure that the node names are unique, the updated ONNX model will be saved in the configured file path. | Default: **None**  
A string. |

### Returns

None

### Outputs

Outputs a quantization configuration file in JSON format. (When quantization is performed again, the existing one output by this API call will be overwritten.)

```json
{
  "version":1,
  "batch_num":2,
  "activation_offset":true,
  "do_fusion":true,
  "skip_fusion_layers":[],
  "layer_name":{
    "quant_enable":true,
    "activation_quant_params":{
      "max_percentile":0.999999,
      "min_percentile":0.999999,
      "search_range":{
```
Example

```python
import amct_onnx as amct

model_file = "resnet101.onnx"
# Create a quantization configuration file.
amct.create_quant_config(config_file="./configs/config.json",
model_file=model_file,
skip_layers=None,
batch_num=1,
activation_offset=True)
```

6.6.2 quantize_model

Description

Quantizes a graph based on the quantization configuration file, inserts the quantization operators, generates a quantization factor record file `record_file`, and returns the calibrated ONNX model.

Prototype

`quantize_model(config_file, model_file, modified_onnx_file, record_file)`

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization configuration file generated by the user, which is used to specify the configuration of the quantization layer in the network.</td>
<td>A string.</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_file</td>
<td>Input</td>
<td>Updated ONNX model generated by the <code>create_quant_config</code> API.</td>
<td>A string.</td>
</tr>
<tr>
<td>modified_onnx_file</td>
<td>Input</td>
<td>File name of the calibrated ONNX model whose data is to be quantized.</td>
<td>A string.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name.</td>
<td>A string.</td>
</tr>
</tbody>
</table>

### Returns

None

### Example

```python
import amct_onnx as amct
model_file = "resnet101.onnx"
scale_offset_record_file = os.path.join(TMP, 'scale_offset_record.txt')
modified_model = os.path.join(TMP, 'modified_model.onnx')
config_file="/configs/config.json"

# Quantize the model.
amct.quantize_model(config_file, model_file, modified_model, scale_offset_record_file)
```

### 6.6.3 save_model

#### Description

Saves an ONNX model file that can be used for both accuracy simulation in the ONNX Runtime environment and inference on the Ascend AI Processor based on the `record_file` quantization factor record file.

#### Restrictions

- This API is called after `batch_num` is reached. Otherwise, the quantization factors are incorrect and the quantization result is compromised.
- This API receives only the ONNX model file returned by 6.6.2 `quantize_model`.
- The quantization factor record file passed to the API call is generated in the 6.6.2 `quantize_model` phase. The factor values will be filled in the model inference phase.

#### Prototype

```
save_model(modified_onnx_file, record_file, save_path)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>modified_onnx_file</td>
<td>Input</td>
<td>File name of the result ONNX model.</td>
<td>A string.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path.</td>
<td>A string.</td>
</tr>
</tbody>
</table>

Returns

None

Outputs

- Outputs a model for accuracy simulation on ONNX Runtime with the name containing the fake_quant keyword.
- Outputs a deployable model file with the name containing the deploy keyword. The model can be deployed on the Ascend AI Processor after being converted by the ATC tool.

When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.

Example

```python
import amct_onnx as amct
# Perform network inference and complete quantization during the inference.
# The calibrated model generated by the 6.6.2 quantize_model API call contains the newly added AMCT custom operators. Therefore, make sure to contain SessionOptions provided by AMCT in the InferenceSession of ONNX Runtime created for inference with the calibration dataset.
for i in batch_num:
onnxruntime.InferenceSession(onnx_model, amct.AMCT_SO).run(None, {'input':input_batch})
# Insert the API call and save the quantized model as an ONNX file.
amct.save_model(modified_onnx_file="./tmp/modified_model.onnx",
record_file="./scale_offset_record_file.txt",
save_path="./results/model")
```

6.6.4 convert_qat_model

Description

Converts an ONNX QAT model to a model of the Ascend format.
Restrictions

The source QAT model must have FakeQuant layers (including QuantizeLinear and DequantizeLinear). Per-channel quantization takes effect on weights only. A QuantizeLinear-DequantizeLinear layer pair must have the same quantization factor.

Prototype

```
convert_qat_model(model_file, save_path, record_file=None)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_file</td>
<td>Input</td>
<td>Path of the .onnx model file to convert.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, ./quantized_model/*model.</td>
<td>A string.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Path of the quantization factor record file (.txt) computed by the user.</td>
<td>A string. Default: None</td>
</tr>
</tbody>
</table>

Returns

None

Outputs

- Outputs a fake-quantized model for testing on CPU/GPU and a deployable model used for ATC conversion.
- (Optional) Outputs a quantization factor record file (.txt), which records the quantization factors of each quantized layer.

Example

```
import amct_onnx as amct
model_file = "./pre_model/mobilenet_v2_qat.onnx"
save_path="./results/model"
amct.convert_qat_model(model_file, save_path)
```

6.7 FAQ
6.7.1 An Error Message Is Displayed During python3-tk Installation

Symptom

When the python3-tk dependency is installed, the following error message is displayed.

```
```

Solution

Copy the missing file `py_compile.py` to the `/usr/lib/python3.7` directory and reinstall the Python.

```
```

Replace `/usr/local/python3.7.5/lib/python3.7/py_compile.py` with the actual path of the file.

6.8 Appendixes

6.8.1 Simplified Post-training Quantization Configuration File

Table 6-20 describes the parameters in the `calibration_config_onnx.proto` template.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required / Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified post-training quantization configuration file of AMCT.</td>
</tr>
<tr>
<td>Message</td>
<td>Required / Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------</td>
<td>----------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>uint32</td>
<td>batch_num</td>
<td>Batch count used for quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>activation_offset</td>
<td>Whether to quantize activations with offset.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_layers</td>
<td>Layers to skip during quantization.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_layer_types</td>
<td>Types of layers to skip during quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>CalibrationConfig</td>
<td>common_config</td>
<td>Common quantization configuration. If a layer is not overridden by <code>override_layer_types</code> or <code>override_layer_configs</code>, this configuration is used.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>override_layer_type</td>
<td>override_layer_types</td>
<td>Type of layers to override.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>override_layer_type</td>
<td>override_layer_configs</td>
<td>Layer to override.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>do_fusion</td>
<td>BN fusion switch. Defaults to <code>true</code>.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_fusion_layer</td>
<td>Layers to skip in BN fusion.</td>
</tr>
<tr>
<td>OverrideLayerType</td>
<td>Required</td>
<td>string</td>
<td>layer_type</td>
<td>Type of layers to quantize.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CalibrationConfig</td>
<td>calibration_config</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td>OverrideLayer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization configuration overriding by layer.</td>
</tr>
<tr>
<td>Message</td>
<td>Required / Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------</td>
<td>---------</td>
<td>----------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Required</td>
<td>string</td>
<td>layer_name</td>
<td></td>
<td>Layer to override.</td>
</tr>
<tr>
<td>Required</td>
<td>CalibrationConfig</td>
<td>calibrati on_conf i g</td>
<td></td>
<td>Quantization configuration to apply.</td>
</tr>
</tbody>
</table>

**Calibration Config**  
- `-` `-` `-` Calibration-based quantization configuration.  
- `-` ARQuQuantize arq_quantize Weight quantization algorithm. **arq_quantize**: ARQ algorithm configuration.  
- `-` FMRQQuantize ifmr_quantize Activation quantization algorithm. **ifmr_quantize**: IFMR algorithm configuration.  

**ARQuantize**  
- `-` `-` `-` ARQ quantization algorithm configuration.  
- Option channel _wise Whether to use different quantization factors for each channel.  

**FMRQuantize**  
- `-` `-` `-` FMR quantization algorithm configuration.  
- Option search_range_start Quantization factor search start.  
- Option search_range_end Quantization factor search end.  
- Option search_step Quantization factor search step.  
- Option max_percentile Upper search limit.
<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>float</td>
<td>min_percentile</td>
<td>Lower search limit</td>
<td></td>
</tr>
</tbody>
</table>

The following is an example simplified quantization configuration file (*quant.cfg*).

```plaintext
# Global quantization parameters
batch_num : 2
activation_offset : true
skip_layers : "layer1.1.conv2"
skip_layer_types:"Conv"
do_fusion: true
skip_fusion_layers : "layer1.1.conv2"
common_config : {
arq_quantize : {
    channel_wise : true
}
ifmr_quantize : {
    search_range_start : 0.7
    search_range_end : 1.3
    search_step : 0.01
    max_percentile : 0.999999
    min_percentile : 0.999999
}
}
override_layer_types : {
    layer_type : "Gemm"
calibration_config : {
arq_quantize : {
    channel_wise : false
}
ifmr_quantize : {
    search_range_start : 0.8
    search_range_end : 1.2
    search_step : 0.02
    max_percentile : 0.999999
    min_percentile : 0.999999
}
}
}
override_layer_configs : {
    layer_name : "layer1.2.conv2"
calibration_config : {
arq_quantize : {
    channel_wise : true
}
ifmr_quantize : {
    search_range_start : 0.8
    search_range_end : 1.2
    search_step : 0.02
    max_percentile : 0.999999
    min_percentile : 0.999999
}
}
```
6.8.2 Quantization Factor Record File

Prototype

The quantization factor record file is a serialized data structure file based on Protobuf. The corresponding Protobuf prototype is defined as follows (you can also find it in `/amct_onnx/proto/scale_offset_record_onnx.proto` under the AMCT installation path):

```protobuf
syntax = "proto2";
package AMCTONNXX;

message SingleLayerRecord {
  optional float scale_d = 1;
  optional int32 offset_d = 2;
  repeated float scale_w = 3;
  repeated int32 offset_w = 4;
  repeated uint32 shift_bit = 5;
  optional bool skip_fusion = 6 [default = true];
}

message MapFiledEntry {
  optional string key = 1;
  optional SingleLayerRecord value = 2;
}

message ScaleOffsetRecord {
  repeated MapFiledEntry record = 1;
}
```

The parameters are described as follows.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleLayerRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization factors for quantization.</td>
</tr>
<tr>
<td>Optional float</td>
<td>scale_d</td>
<td></td>
<td></td>
<td>Scale factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td>Optional int32</td>
<td>offset_d</td>
<td></td>
<td></td>
<td>Offset factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td>Message</td>
<td>Required / Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------</td>
<td>-------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ScaleOffset</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Map structure. To ensure compatibility, the discrete map structure is used.</td>
</tr>
<tr>
<td>Record</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Repeat</td>
<td>float</td>
<td>scale_w</td>
<td>Scale factor for weight quantization. Scalar mode (quantizing the weight of the current layer in a unified manner) and vector mode (quantizing the weight of the current layer in channel-wise mode) are supported. Only Conv2d layers support channel-wise quantization.</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>int32</td>
<td>offset_w</td>
<td>Offset factor for weight quantization. Similar to scale_w, it also supports the scalar and vector quantization modes and the dimension configuration must be the same as that of scale_w. Currently weight quantization with offset is not supported, and offset_w must be 0.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>skip_fusion</td>
<td>Whether to skip Conv+BN fusion at the current layer. Defaults to false, indicating that the preceding fusion type is performed.</td>
<td></td>
</tr>
<tr>
<td>MapFiledEntry</td>
<td></td>
<td>record</td>
<td>Each records a quantization factor of a quantization layer and consists of two members:</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>string</td>
<td>key</td>
<td>Layer name.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>SingleLayerRecord</td>
<td>value</td>
<td>Quantization factor configuration.</td>
<td></td>
</tr>
</tbody>
</table>
Beware that the Protobuf protocol does not report an error for repeated settings of optional fields. Instead, the most recent settings are used.

For general quantization layers, a range of parameters need to be configured, including `scale_d`, `offset_d`, `scale_w`, `offset_w`, and `shift_bit`.

```protobuf
record {
  key: "conv1"
  value {
    scale_d: 0.0798481479
    offset_d: 1
    scale_w: 0.00297622895
    offset_w: 0
    skip_fusion: true
  }
}
record {
  key: "layer1.0.conv1"
  value {
    scale_d: 0.00392156886
    offset_d: -128
    scale_w: 0.00106807391
    scale_w: 0.00104224426
    scale_w: 0.0010603976
    offset_w: 0
    offset_w: 0
    offset_w: 0
  }
}
```

**Quantization Factors**

The scale (a floating-point number) and offset quantization factors need to be provided for data and weight quantization. AMCT uses a unified quantization data structure. See the following expression.

\[
data_{\text{int8}} = \text{clip}_{\text{int8}}\left(\text{round}\left(\frac{data_{\text{float}}}{\text{scale}} + \text{offset}\right)\right)
\]

The value ranges are as follows:

- \( \text{scale} \in \left[\text{FLT\_EPSILON}, \frac{1}{\text{FLT\_EPSILON}}\right] \), \( \text{FLT\_EPSILON} \approx 1.1920929 \times 10^{-7} \)
- \( \text{offset} \in [-128, 127] \)

Quantization algorithms are classified into symmetric quantization algorithms and asymmetric quantization algorithms.

1. **Symmetric quantization algorithm:**
   
   The original high-precision data and quantized int8 data are converted into
   \( data_{\text{float}} = \text{scale} \times data_{\text{int8}} \), where `scale` is a float32. To indicate positive and negative numbers, the signed int8 data type is used for
   \( data_{\text{float}} = \text{scale} \times data_{\text{int8}} \). The following describes how to convert the original data into the int8 format. `round` is a rounding function. The value to be determined by the quantization algorithm is the constant `scale`.

   \[
data_{\text{int8}} = \text{round}\left(\frac{data_{\text{float}}}{\text{scale}}\right)
\]
Quantization of the weights and activations may be summarized as a process of searching for a scale. Because data\_{int8} is a signed number, to ensure symmetry of the ranges represented by positive and negative values, an absolute value operation is first performed on all data, so that the range of the to-be-quantized data is changed to data\_{int8}, and then scale is determined. The range of positive int8 values is [0, 127]. Therefore, scale can be computed as follows:

\[
\text{scale} = \frac{\text{data}_{\text{max}}}{127}
\]

Therefore, the range of the int8 values is \([-128 \times \text{scale}, 127 \times \text{scale}]\). Data beyond the range \([-128 \times \text{scale}, 127 \times \text{scale}]\) is saturated to a boundary value, and then the quantization operation shown in the formula is performed.

2. Asymmetric quantization algorithm:

The difference from symmetric quantization algorithms lies in the data conversion mode. The scale and offset constants also need to be determined.

\[
data_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset})
\]

The uint8 data is converted through calculation based on the original high-accuracy data, which is shown in the following formula:

\[
data_{\text{uint8}} = \text{round} \left( \frac{\text{data}_{\text{float}}}{\text{scale}} - \text{offset} \right)
\]

scale is an fp32, data\_{uint8} is an unsigned int8, and offset is an int8. The data range is \([\text{scale} \times \text{offset}, \text{scale} \times (255 + \text{offset})]\) if a value range of the to-be-quantized data is \([\text{data}_{\text{min}}, \text{data}_{\text{max}}]\). scale and offset are computed as follows:

\[
\text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}, \quad \text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}
\]

Unified quantization data format: By performing simple data conversion of the asymmetric quantization algorithm formula, the quantized data and the symmetric quantization algorithm are of the same type, that is, int. The specific conversion process is as follows:

int8 quantization is used as an example for description. The input original floating-point data is data\_{float}, the original quantized fixed-point number is data\_{float}, the quantization scale is scale, and the quantization offset is offset (the algorithm requires zero crossing to avoid too much accuracy drop). The calculation principle of quantization is as follows:

\[
data_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset}) = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset} + 128) = \text{scale} \times (\text{data}_{\text{uint8}} - \text{offset}')
\]

Where,

\[
data_{\text{int8}} = \text{data}_{\text{uint8}} - 128 \in [-128, 127], \quad \text{offset}' = -(\text{offset} + 128) \in [-128, 127]
\]

Through the foregoing conversion, the data may also be converted into the int8 format. After scale and the converted offset' are determined, the int8 data converted from the original floating-point data is as follows:
6.8.3 Python 3.7.5 Installation on Ubuntu

**Step 1** Check that the Python 3.7.5 development environment is available.

Run the `python3.7.5 --version`, `python3.7 --version`, `pip3.7.5 --version`, and `pip3.7 --version` commands to check whether the environment is available. If the following information is displayed, the environment is available. Otherwise, go to the next step.

| Python 3.7.5 | pip 19.2.3 from /usr/local/python3.7.5/lib/python3.7/site-packages/pip (python 3.7) |

**Step 2** Install the Python 3.7.5 dependencies.

- `sudo apt-get install -y make zlib1g zlib1g-dev build-essential libbz2-dev libsqlite3-dev libssl-dev libxml2-dev libffi-dev openssl python3-tk`

`libsqlite3-dev` must be installed before the Python installation. If the Python 3.7.5 environment has been installed in the user’s OS before the `libsqlite3-dev` installation, you need to rebuild the Python environment. If `python3-tk` fails to be installed, see 6.7.1 An Error Message Is Displayed During `python3-tk` Installation.

**Step 3** Install Python 3.7.5.

1. Run the `wget` command to download the source package of Python 3.7.5 to any directory on the server where AMCT is located:
   ```
   wget https://www.python.org/ftp/python/3.7.5/Python-3.7.5.tgz
   ```

2. Go to the download directory and decompress the source package:
   ```
   tar -zxvf Python-3.7.5.tgz
   ```

3. Go to the new folder and run the following configuration, build, and installation commands:
   ```
   cd Python-3.7.5
   ./configure --prefix=/usr/local/python3.7.5 --enable-loadable-sqlite-extensions --enable-shared
   make
   sudo make install
   ```

   --prefix specifies the Python installation path. You can modify it as required.
   --enable-shared is used to build the `libpython3.7m.so.1.0` dynamic library. --enable-loadable-sqlite-extensions is used to load the sqlite-devel dependency.

   This document uses --prefix=/usr/local/python3.7.5 as an example. After the configuration, build, and installation commands are executed, the installation package is output to the /usr/local/python3.7.5 directory, and the `libpython3.7m.so.1.0` dynamic library is output to the /usr/local/python3.7.5/lib/libpython3.7m.so.1.0 directory.

4. Set the soft links:
   ```
   sudo ln -s /usr/local/python3.7.5/bin/python3 /usr/local/python3.7.5/bin/python3.7.5
   sudo ln -s /usr/local/python3.7.5/bin/pip3 /usr/local/python3.7.5/bin/pip3.7.5
   ```

5. Set the Python 3.7.5 environment variables.
   a. If Python is installed by the root user:
      AMCT is installed by the root user. Run the following commands in the current terminal window to set environment variables:
# Set the Python 3.7.5 library path.
export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
# If multiple Python 3 versions exist in the user environment, specify Python 3.7.5.
export PATH=/usr/local/python3.7.5/bin:$PATH

**NOTICE**

If the running user is the root user, it is not advised to modify the .bashrc file. Otherwise, the Python tools provided by other systems may be unavailable. If you want to use the default tool, open another terminal window.

b. If Python is installed by a non-root user:

AMCT is also installed by the non-root user. Run the `vi ~/.bashrc` command in any directory to open the .bashrc file, and append the following lines to the file:

```bash
# Set the Python 3.7.5 library path.
export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
# If multiple Python 3 versions exist in the user environment, specify Python 3.7.5.
export PATH=/usr/local/python3.7.5/bin:$PATH
```

Run the `:wq!` command to save the file and exit. Run the `source ~/.bashrc` command for the modification to take effect immediately.

**Step 4** After the installation is complete, run the following commands to check the installed version. If the required version information is displayed, the installation is successful.

```bash
python3.7.5 --version
pip3.7.5 --version
python3.7 --version
pip3.7 --version
```

---End
7 AMCT Instructions (MindSpore)

7.1 Introduction

7.1.1 Overview

This document describes how to quantize a MindSpore model using Ascend Model Compression Toolkit (AMCT). In the quantization process, the precision of model weights and activations is reduced to make the model more compact, improving the compute efficiency while lowering the transfer latency.

AMCT is a MindSpore-based Python toolkit that implements Conv+BN fusion as well as 8-bit quantization of data and weights in neural networks. This toolkit decouples quantization from model conversion. It implements independent quantization of quantization-capable layers in a model and saves the quantized model to an .air file. The obtained accuracy simulation model can run on NPU to complete accuracy simulation. The obtained deployable model can run on the Ascend AI Processor with improved inference performance. The MindSpore model quantized by AMCT can serve for both accuracy simulation and inference deployment. This tool has the following advantages:

- Lightweight: You only need to install the tool.
- Easy-to-use APIs: You can complete quantization using APIs based on the MindSpore inference script.
- Hardware compatibility: The quantized model can be converted to an offline model by using the ATC tool to implement 8-bit inference on the Ascend AI Processor.
- Configurable quantization: You can modify the quantization configuration file and adjust the quantization strategy to obtain the optimal quantization result.

Figure 7-1 shows the AMCT architecture.

7.1.2 Features

7.1.2.1 Terminology

There are two forms of quantization: post-training quantization and quantization aware training. The foregoing two quantization forms are classified into weight quantization and activation quantization according to the quantization object.

As used in this document, the following terms have the meanings specified below.

Post-training Quantization

Post-training quantization refers to quantizing the weights of an already-trained model from float32 to int8 and calibrating and quantizing the activations by using a small calibration dataset. For details about the quantization workflow, see 7.3 Post-training Quantization.

- Calibration dataset
  During the calibration process, the algorithm uses each piece of data in the calibration dataset as input, accumulates the input data of each layer (or operator) that needs to be quantized, and uses the accumulated data as input data of the quantization algorithm to determine the quantization factors. Because the quantization factors and the accuracy of the quantized model are closely related to the selection of the calibration dataset, you are advised to calibrate the model using a subset of images from the validation dataset.
• **Activation quantization**
  Activation quantization is to collect statistics on the input data of each layer (or operator) to be quantized, to find the optimal pair of scale and offset per layer (or operator). For details, see 7.8.3 **Quantization Factor Record File**.

Activations are the intermediate results of model inference computation. The value ranges are input-specific. Therefore, a group of reference inputs (a calibration dataset) needs to be used as incentives to record the input data of each layer (or operator) to be quantized for searching for quantization factors (scale and offset).

• **Weight quantization**
  After model training, the weights and the value ranges are determined. Therefore, quantization may be directly performed based on the value range of each weight.

The layers that support post-training quantization are listed as follows.

**Table 7-1** Layers that support post-training quantization and restrictions

<table>
<thead>
<tr>
<th>Layer</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>nn.Dense</td>
<td>-</td>
</tr>
<tr>
<td>nn.Conv2d</td>
<td>Using a 1-dilated 4 x 4 filter with <code>group = 1</code>, or a 1-dilated 4 x 4 filter with <code>group = channel ≠ 1</code></td>
</tr>
</tbody>
</table>

**Quantization Aware Training**

Quantization aware training introduces quantization in the forward pass of the training process, allowing for higher accuracy.

Distributed training can be implemented by using a plurality of devices. Therefore, quantization aware training can be classified into quantization aware training using a single device and quantization aware training using multiple devices (distributed training).

Quantization aware training is time consuming and data hungry. For details about the quantization workflow, see 7.4 **Quantization Aware Training**.

• **Training dataset**
  Dataset of the already-trained network.

• **Activation quantization**
  Activation quantization refers to iterative training of the upper clip limit and lower clip limit, which are used to calculate the current scale and offset. The activation is the intermediate result of model inference and calculation. The ULQ retrain algorithm is used to continuously optimize the two factors during the quantization aware training process to obtain the optimal factors.

• **Weight quantization**
  Weight quantization means to optimize the quantization parameters of weights during the quantization aware training process to obtain the optimal parameters.
The layers that support quantization aware training are listed as follows.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>nn.Dense</td>
<td>-</td>
</tr>
<tr>
<td>nn.Conv2d</td>
<td>Using a 1-dilated 4 x 4 filter with group = 1, or a 1-dilated 4 x 4 filter with group = channel ≠ 1</td>
</tr>
</tbody>
</table>

### 7.1.2.2 Principles

Figure 7-2 shows the AMCT principles. The blue parts are implemented by the user, and the gray part is implemented by the AMCT APIs called by the user.

- **Post-training Quantization**
  a. Construct the original MindSpore model and then call the 7.7.2.1 create_quant_config API to generate a quantization configuration file.
  b. Call the 7.7.2.2 quantize_model API to optimize the original MindSpore model based on the quantization configuration file. The optimized model contains operators of the quantization algorithm. Then, perform inference with the optimized model in the MindSpore environment based on the downloaded image dataset and calibration dataset to obtain the quantization factors. The calibration dataset used to generate quantization factors must be obtained from the training dataset or test dataset to keep data distribution as close to the actual data as possible for better inference accuracy of the quantized model.
  c. Call the 7.7.2.3 save_model API to save the quantized model file in AIR format, which is deployable on the Ascend AI Processor.

- **Quantization Aware Training**
  a. Construct an original MindSpore model and call 7.7.3.1 create_quant_retrain_config to generate a quantization aware training configuration file. 7.3 Post-training Quantization is performed in the quantization aware training process and a calibration_record.txt file is generated.
  b. Perform 7.7.3.2 UlqInitializer with the obtained calibration_record.txt file, and call the 7.7.3.3 create_quant_retrain_model API to modify the original MindSpore model. The modified model contains operators of the quantization aware training algorithm.
  c. If the training process is not interrupted, call 7.7.3.5 save_quant_retrain_model to save the model file in AIR format after quantization aware training, which is deployable on the Ascend AI Processor.

  If the training process is interrupted, call 7.7.3.4 restore_quant_retrain_model again to output the modified retrained network based on the saved checkpoint and quantization configuration files and continue quantization-aware training. After the training is
complete, call **7.7.3.5 save_quant_retrain_model** to save the quantized model in AIR format.

**Figure 7-2** Tool principles

![Tool principles diagram]

### 7.1.3 Fusion Support

Currently, this tool implements the following form of BN fusion:

Conv+BN fusion: Before AMCT quantization, the "nn.Conv2d+nn.BatchNorm2d" composite in the model is fused into "Conv+BN." After fusion, the nn.BatchNorm2d layer is removed.

### 7.1.4 Tool Workflow

**Figure 7-3** shows the tool workflow.
Figure 7-3 Tool workflow

Set up Ascend 910 environment

Install MindSpore

Install AMCT

Set environment variables

Run quantization

Run inference with the quantized model?

Table 7-3 Major actions in the tool workflow

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set up Ascend 910 environment</td>
<td>Set up the Ascend 910 environment and install components such as Driver, Firmware, and AI CPU by referring to CANN Software Installation Guide of Ascend 910 AI Processor.</td>
</tr>
<tr>
<td>Install MindSpore</td>
<td>Click here to install MindSpore of the required version and verify the installation.</td>
</tr>
<tr>
<td>Install AMCT</td>
<td>Install the MindSpore AMCT by referring to 7.2 AMCT Installation. Before the installation, you need to obtain the AMCT package, create an AMCT user, check the environment, install dependencies, and upload the AMCT package.</td>
</tr>
</tbody>
</table>
| Run quantization | 1. Set environment variables by referring to 7.2.4 Post-installation Actions.  
2. Prepare the model, image dataset, and calibration dataset, and use the AMCT to quantize your MindSpore model. For details, see 7.3 Post-training Quantization and 7.4 Quantization Aware Training. |
<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
</table>
| (Optional) Run inference on the quantized model | 1. Set up an inference environment (the scenario where the development environment and operating environment are co-deployed is recommended) by referring to *CANN Software Installation Guide of Ascend 310 AI Processor* (inference purposes) and upload the quantized model to the environment.  
2. Use the ATC tool to convert the quantized model by referring to *ATC Instructions* of Ascend 310 AI Processor (inference purposes), and then run inference on the converted model. |

### 7.2 AMCT Installation

#### 7.2.1 获取软件包

#### 7.2.2 Pre-installation Actions

Before quantizing and converting a MindSpore model, prepare the environment as follows and then install AMCT.

**Preparing the AMCT User**

Perform the installation as the user who has installed the components in the Ascend 910 environment. This section uses a non-root user as an example.

For details about the permission settings of a non-root user, see section "Pre-installation Actions" in *CANN Software Installation Guide of Ascend 910 AI Processor*.

**Setting Up Environment**

**NOTE**

This tool supports Python 3.7.0 to Python 3.7.9. This document uses Python 3.7.5 as an example. The environment variables and installation commands vary according to the actual Python version.

AMCT needs to be installed in the Ascend 910 environment, which has the following hardware requirements.
### Table 7-4 Hardware requirements

<table>
<thead>
<tr>
<th>Item</th>
<th>Version</th>
<th>How to Obtain</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ascend 910 OS</td>
<td>Ubuntu 18.04 (x86_64), EulerOS release 2.0 (SP9) AArch64</td>
<td>See <em>CANN Software Installation Guide</em> of Ascend 910 AI Processor.</td>
<td>-</td>
</tr>
<tr>
<td>Python</td>
<td>3.7.5</td>
<td>Install the dependencies in both the Ascend 910 and Ascend 310 environments by referring to the &quot;Pre-installation Actions&quot; section of the corresponding OS in <em>CANN Software Installation Guide</em></td>
<td>-</td>
</tr>
<tr>
<td>MindSpore</td>
<td>1.2.0</td>
<td>Available at: <a href="https://www.mindspore.cn/install/en">https://www.mindspore.cn/install/en</a></td>
<td>After the installation is complete, set environment variables and verify the installation.</td>
</tr>
<tr>
<td>Ascend 310 OS</td>
<td>Ubuntu 18.04 (x86_64), EulerOS release 2.0 (SP9) AArch64</td>
<td>See <em>CANN Software Installation Guide</em> of Ascend 310 AI Processor. This environment is required for converting the quantized model into an offline model adapted to the Ascend AI Processor and running inference with the offline model.</td>
<td>-</td>
</tr>
</tbody>
</table>

### Installing Dependencies

For details, see “Pre-installation Actions” in *CANN Software Installation Guide* of Ascend 910 AI Processor. After the installation is complete, run the following command to install the wget software, which is required for downloading model files:

```
Run the `pip3.7.5 list` command to check whether the wget software is installed. If the version information is displayed, wget has been installed. Otherwise, run the following command to install wget:
```

```bash
```
pip3.7.5 install wget==3.2 --user

### Uploading the AMCT Package

Upload the `Ascend-amct-{software version}-{os}.{arch}.tar.gz` package to any directory on the Linux server as the AMCT installation user. In this example, the package is uploaded to `$HOME/amct/`.

Decompress the AMCT package:

```bash
tar -zxvf Ascend-amct-{software version}-{os}.{arch}.tar.gz
```

Find the following extracted packages.

**Table 7-5 Extracted parts of the AMCT package**

<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>amct_caffe</code>/</td>
<td>Caffe AMCT directory</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
|                   | `amct_caffe-{version}-py3-none-linux_{arch}.whl` | Caffe AMCT package | • OS support: Ubuntu 18.04 (x86_64)  
|                   | `amct_caffe_sample.tar.gz` | Caffe quantization sample package | • For details, see AMCT Instructions (Caffe).  
|                   | `caffe_patch.tar.gz` | Caffe patch package | • Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| `amct_tensorflow`/ | TensorFlow AMCT directory |             |                         |
|                   | `amct_tensorflow-{version}-py3-none-linux_{arch}.whl` | TensorFlow AMCT package | • OS support: Ubuntu 18.04 (x86_64)  
|                   | `amct_tensorflow_sample.tar.gz` | TensorFlow quantization sample package | • `amct_tensorflow` and `amct_tensorflow_ascend` cannot exist at the same time.  
|                   | `amct_tensorflow_ascend-{version}-py3-none-linux_{arch}.whl` | TF Adapter AMCT package | • For details, see AMCT Instructions (TensorFlow).  
|                   | | | • Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor.  
<p>|                   | | | • OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment |</p>
<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>amct_tensorflow</strong></td>
<td><strong>amct_tensorflow_ascend_sample.tar.gz</strong></td>
<td>Package of quantization samples using TF Adapter</td>
<td></td>
</tr>
</tbody>
</table>
  - *amct_tensorflow* and  
    *amct_tensorflow_ascend* cannot exist at the same time.  
  - For details, see *AMCT Instructions (TensorFlow, Ascend)*.  
  - Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| **amct_pytorch/** | **PyTorch AMCT directory** |  
  - *amct_pytorch-{version}-py3-none-linux_{arch}.tar.gz*  
    PyTorch AMCT source package  
  - *amct_pytorch_sample.tar.gz*  
    PyTorch quantization sample package |  
  - OS support: Ubuntu 18.04 (x86_64)  
  - For details, see *AMCT Instructions (PyTorch)*.  
  - Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| **amct_onnx/** | **ONNX AMCT directory** |  
  - *amct_onnx-{version}-py3-none-linux_{arch}.whl*  
    ONNX AMCT package  
  - *amct_onnx_op.tar.gz*  
    ONNX Runtime AMCT custom OPP  
  - *amct_onnx_sample.tar.gz*  
    ONNX quantization sample package |  
  - OS support: Ubuntu 18.04 (x86_64)  
  - For details, see *AMCT Instructions (ONNX)*.  
  - Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| **amct_mindspore/** | **MindSpore AMCT directory** |  
  - *amct_mindspore-{version}-py3-none-linux_{arch}.whl*  
    MindSpore AMCT package |  
  - OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
  - For details, see *AMCT Instructions (MindSpore)*. |
### 7.2.3 Installation

**Step 1** In `amct/amct_ms` under the directory where the AMCT package is located, run the following command:

```bash
pip3.7.5 install amct_mindspore-{version}-py3-none-linux_{arch}.whl --user
```

Replace `{version}` with the actual AMCT version number, and `{arch}` with the actual architecture of the installation server. If AMCT installation is performed by the root user and the `--target` option is included, ensure that the path specified by `--target` is the path of the current user.

**Step 2** Check the installation. If a message similar to the following is displayed, the installation is successful:

```
Successfully installed amct-mindspore-{version}
```

Find the installed AMCT in the `python3.7.5` directory (for example, `$HOME/.local/lib/python3.7.5/site-packages`).

```
drwx------ 15 amct amct   4096 Mar 17 11:50 amct_mindspore/
drwx------  2 amct amct   4096 Mar 17 11:50 amct_mindspore-{version}.dist-info/
```

*amct_mindspore* indicates the AMCT installation path.

-----End

### 7.2.4 Post-installation Actions

After the components are installed for Ascend 910 AI Processor and before you start quantization, set the component environment variables. The following
example assumes that Driver is installed by the root user to the default installation path (/usr/local/Ascend) and FwkACLlib and OPP are installed by a non-root user to the default installation path (SHOME/Ascend).

```bash
# Library path
export LD_LIBRARY_PATH=/usr/local/Ascend/add-ons:${install_path}/fwkacllib/lib64:${LD_LIBRARY_PATH}

# TBE operator execution path
export TBE_IMPL_PATH=${install_path}/opp/op_impl/built-in/ai_core/tbe

# Compiler and component execution path
export PATH=${install_path}/fwkacllib/ccec_compiler/bin:${install_path}/fwkacllib/bin:${install_path}/toolkit/bin:$PATH

# Component dependency path
export PYTHONPATH=${TBE_IMPL_PATH}:${install_path}/fwkacllib/python/site-packages:${install_path}/tfplugin/python/site-packages:${install_path}/toolkit/python/site-packages:$PYTHONPATH

# Weights and activation quantization factors generation variable. When the 7.7.2.3 save_model API is called, an amct_dump directory is created in the script directory, and a record.txt file is generated to the directory to record weights and activation quantization factors. This environment variable is required if 7.4 Quantization Aware Training will be performed.
export DUMP_AMCT_RECORD=1
```

You can choose to modify the ~/.bashrc file to set permanent environment variables. The following uses bash shell as an example:

1. Run the vi ~/.bashrc command in any directory as the installation user to open the .bashrc file and append the preceding lines to the file.
2. Run the :wq! command to save the file and exit.
3. Run the source ~/.bashrc command for the modification to take effect immediately.

### 7.3 Post-training Quantization

#### 7.3.1 Prerequisites

**Model**

1. Click here to download the pre-trained resnet50.ckpt file of ResNet-50.
2. Upload the downloaded model to any directory on the AMCT server, for example, the SHOME/AMCT/model directory.

**Image Dataset**

1. Click here to download the CIFAR-10 dataset corresponding to the ResNet-50. Click CIFAR-10 binary version (suitable for C programs) in the Download area.
2. Upload the downloaded dataset to any directory on the AMCT server, for example, the SHOME/AMCT/dataset directory.
3. Go to the directory and extract the package:
   ```bash
tar -zxvf cifar-10-binary.tar.gz
```
   Find the extracted cifar-10-batches-bin directory.
Calibration Dataset

Create the `cifar-10-verify-bin` calibration dataset directory in the parent directory of the `cifar-10-batches-bin` directory, and copy the following files in the `cifar-10-batches-bin` directory to the created directory.

```bash
mkdir cifar-10-verify-bin
cp cifar-10-batches-bin/batches.meta.txt ./cifar-10-verify-bin/
cp cifar-10-batches-bin/readme.html ./cifar-10-verify-bin/
cp cifar-10-batches-bin/test_batch.bin ./cifar-10-verify-bin/
```

7.3.2 Quantization Example

Procedure

1. Go to the `amct/amct_ms` directory and decompress the sample package:

   ```bash
tar -zxvf amct_mindspore_sample.tar.gz
cd sample/ResNet50
```

   Find the following extracted files and directories:
   - `download_files.py`: the script that downloads model definition file and performs dataset processing.
   - `resnet50_sample.py`: post-training quantization script.
   - `resnet50_retrain_sample.py`: single-device quantization aware training script. For details about the quantization method, see 7.4 Quantization Aware Training.
   - `README.md`: description of the quantization script.
   - `src/retrain_config.py`: configuration script for quantization aware training. This script is invoked during quantization aware training to control training parameters such as the learning rate.
   - `scripts/run_distribute_train_sample.sh`: distributed quantization aware training script. For details about the quantization method, see 7.4.2 Multi-Device Quantization Example.
   - `scripts/run_standalone_train_sample.sh`: encapsulated script for single-device quantization aware training.

2. Download the model files.

   Go to the `sample/ResNet50` directory and run the following command to download the model definition file:

   ```bash
   python3.7.5 download_files.py --close_certificate_verify
   ```

   The `--close_certificate_verify` option is optional. It is used to disable certificate verification to ensure that the model can be downloaded properly. If a message is displayed indicating that authentication fails during model download, include this option to your download command and try again.

   If messages similar to the following are displayed, the download is successful:

   ```
   # Model definition file. The file is instantiated during quantization to generate a model file.
   # Data preprocessing script. This script is called to process the dataset when the quantization script is executed.
   # Script for controlling the learning rate. This script is called during training to control the learning rate.
   # Script used in the distributed training quantization scenario, which is used to generate the HCCL configuration file.
   ```
3. Quantize the ResNet-50 network model:

   `python3.7.5 resnet50_sample.py --dataset_path your dataset path --checkpoint_path your resnet50 checkpoint file path`

   The command-line options are described as follows.

   **Table 7-6 Command-line options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required / Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Displays help information.</td>
<td>Optional</td>
</tr>
<tr>
<td>--dataset_path</td>
<td>Sets the dataset directory, for example, $HOME/AMCT/dataset.</td>
<td>Required</td>
</tr>
<tr>
<td>DATASET_PATH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--checkpoint_path</td>
<td>Sets the directory of the MindSpore weight file resnet50.ckpt.</td>
<td>Required</td>
</tr>
<tr>
<td>CHECKPOINT_PATH</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   An example is as follows.

   `python3.7.5 resnet50_sample.py --dataset_path $HOME/AMCT/dataset/cifar-10-verify-bin --checkpoint_path $HOME/AMCT/model/resnet50.ckpt`

4. If messages similar to the following are displayed, the quantization is successful:

   ```
   INFO - [AMCT]:[QuantizeTool]: Generate AIR file: $HOME/amct/amct_mindspore/sample/ResNet50/results/resnet50_quant.air success!
   INFO] the quantized AIR file has been stored at: results/resnet50_quant.air
   ```

5. View the quantization result.

   After successful quantization, find the following files generated to the sample/ResNet50 directory:

   - **config.json**: configuration file that describes how to quantize each layer in the model. If a quantization configuration file already exists in the directory of the quantization script, the existing quantization configuration file is overwritten by a new one with the same name in a call to 7.7.2.1 `create_quant_config`. If not that case, a new quantization configuration file is created.

   If the accuracy of model inference drops significantly after quantization, you can modify the **config.json** file by referring to 7.3.3 Quantization Configuration.

   - **amct_log/amct_mindspore.log**: quantization log file.
   - **results/resnet50_quant.air**: quantization result file resnet50_quant.air, which is deployable on the Ascend AI Processor.
   - **kernel_meta**: directory of operator build outputs.
   - (Optional) **amct_dump/calibration_record.txt**: If the environment variable `export DUMP_AMCT_RECORD=1` has been set in 7.2.4 Post-installation Actions before quantization, a quantization factor directory containing the quantization factor record file **calibration_record.txt** is generated in the directory of the quantization script. For details about the
prototype definition of the file, see 7.8.3 Quantization Factor Record File.

6. (Optional) Convert the quantized model into an offline model adapted to the Ascend AI Processor.
   a. Set up the inference environment by referring to CANN Software Installation Guide of Ascend 310 AI Processor.
   b. Upload the quantized model to the environment.
   c. Convert the model by referring to ATC Instructions of Ascend 310 AI Processor.

7.3.3 Quantization Configuration

This chapter uses the quantization configuration file of a classification network as an example.

7.3.3.1 Overview

If inference based on the config.json post-training quantization configuration file generated by the call to 7.7.2.1 create_quant_config has significant accuracy drop, tune the config.json file until the accuracy is as expected. The following is an example of the file content. Keep the layer names unique in the JSON file.

```json
{
    "version":1,
    "activation_offset":true,
    "do_fusion":true,
    "skip_fusion_layers":[]

    "conv1":{
        "quant_enable":true,
        "activation_quant_params":{
            "max_percentile":0.999999,
            "min_percentile":0.999999,
            "search_range":[0.7, 1.3],
            "search_step":0.01
        },
        "weight_quant_params":{
            "channel_wise":true
        }
    }

    "end_point":{
        "quant_enable":true,
        "activation_quant_params":{
            "max_percentile":0.999999,
            "min_percentile":0.999999,
            "search_range":[0.7, 1.3],
            "search_step":0.01
        },
        "weight_quant_params":{
            "channel_wise":false
        }
    }

    "layer1.0.conv1":{
        "quant_enable":true,
        "activation_quant_params":{
            "max_percentile":0.999999,
            "min_percentile":0.999999,
            "search_range":[]
        }
    }
}
```
0.7,
  1.3
},
  "search_step":0.01
},
  "weight_quant_params":{
    "channel_wise":true
  }
},
"layer1.0.conv2":{
  "quant_enable":true,
  "activation_quant_params":{
    "max_percentile":0.999999,
    "min_percentile":0.999999,
    "search_range":[
      0.7,
      1.3
    ],
    "search_step":0.01
  },
  "weight_quant_params":{
    "channel_wise":true
  }
}

7.3.3.2 Configuration File Options

The following tables describe the parameters in the configuration file.

Table 7-7 version

<table>
<thead>
<tr>
<th>Function</th>
<th>Version number of the quantization configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>Currently, only version 1 is available.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>1</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 7-8 activation_offset

<table>
<thead>
<tr>
<th>Function</th>
<th>Symmetric quantization or asymmetric quantization select for activation quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
<tr>
<td>Description</td>
<td>If it is set to true, asymmetric quantization is used. If it is set to false, symmetric quantization is used.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>true</td>
</tr>
<tr>
<td>------------------------</td>
<td>------</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 7-9 do_fusion**

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th>Fusion switch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>true or false</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If it is set to <strong>true</strong>, fusion is enabled. If it is set to <strong>false</strong>, fusion is disabled. For details about layers that support fusion and fusion patterns, see <strong>7.1.3 Fusion Support</strong>.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 7-10 skip_fusion_layers**

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th>Layer skip in fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>string</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>Layers that support fusion. For details about layers that support fusion and fusion patterns, see <strong>7.1.3 Fusion Support</strong>.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Layers to skip in fusion</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 7-11 layer_config**

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th>Quantization configuration of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>object</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
| **Description** | Includes the following parameters:  
|                 | ● quant_enable  
|                 | ● activation_quant_params  
|                 | ● weight_quant_params |
| **Recommended Value** | None |
| **Required/Optional** | Optional |

**Table 7-12 quant_enable**

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th>Quantization enable by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>true or false</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If it is set to true, the layer is to be quantized. If it is set to false, otherwise.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 7-13 activation_quant_params**

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th>Activation quantization parameters of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>object</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
| **Description** | Includes the following parameters:  
|                 | ● max_percentile  
|                 | ● min_percentile  
|                 | ● search_range  
|                 | ● search_step |
| **Recommended Value** | None                        |
| **Required/Optional** | Optional                 |
### Table 7-14 weight_quant_params

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th>Weight quantization parameters of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>object</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Includes the following parameter: channel_wise</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 7-15 max_percentile

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th>Upper search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>float</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>(0.5, 1]</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Indicates the maximum number to be considered as the search result among a group of numbers in descending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the maximum, that is, the largest number. A larger value indicates that the upper clip limit is closer to the maximum value of the data to be quantized.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>0.999999</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 7-16 min_percentile

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th>Lower search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>float</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>(0.5, 1]</td>
</tr>
</tbody>
</table>
Description Indicates the minimum number to be considered as the search result among a group of numbers in ascending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the minimum, that is, the smallest number.

A larger value indicates that the lower clip limit is closer to the minimum value of the data to be quantized.

Recommended Value 0.999999

Required/Optional Optional

Table 7-17 search_range

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search range: [search_range_start, search_range_end]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>A list of two floats</td>
</tr>
<tr>
<td>Value Range</td>
<td>0 &lt; search_range_start &lt; search_range_end</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the quantization factor search range:</td>
</tr>
<tr>
<td></td>
<td>• search_range_start: search start.</td>
</tr>
<tr>
<td></td>
<td>• search_range_end: search end.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>[0.7, 1.3]</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 7-18 search_step

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0, (search_range_end – search_range_start)]</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the quantization factor search step. A smaller value indicates a smaller step. The number of search iterations is specified by search_iteration = (search_range_end – search_range_start)/search_step. The search becomes time-consuming with the increase of the number of search iterations. In this scenario, processes may be suspended.</td>
</tr>
</tbody>
</table>
7.3.3.3 Configuration Tuning

If the inference accuracy of the model quantized based on the default config.json configuration drops significantly, perform the following steps to tune the quantization configuration file:

**Step 1** Execute the quantization script in the amct_mindspore_sample.tar.gz package to perform quantization based on the default configuration generated by the 7.7.2.1 create_quant_config API.

**Step 2** If the inference accuracy with the model quantized in **Step 1** is as expected, configuration tuning ends. Otherwise, go to **Step 3**.

**Step 3** To perform inference with the graph modified by 7.7.2.2 quantize_model, appropriately increase the value of batch_size of the dataset. Generally, a larger quantity of data samples used in a quantization process indicates a smaller accuracy loss after quantization. However, excessive data does not improve accuracy, but occupies more memory, reduces the quantization speed, hence resulting in insufficient memory, video RAM, and thread resources. You are advised to set batch_size to 16 or 32.

**Step 4** If the inference accuracy with the model quantized in **Step 3** is as expected, configuration tuning ends. Otherwise, go to **Step 5**.

**Step 5** Manually tune the value of quant_enable in the quantization configuration file.

---

Table 7-19 channel_wise

<table>
<thead>
<tr>
<th>Function</th>
<th>Whether to use different quantization factors for each channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td>true or false</td>
</tr>
</tbody>
</table>
| Description | ● If it is set to **true**, channels are separately quantized using different quantization factors.  
            ● If it is set to **false**, all channels are quantized altogether using the same quantization factors. |
| NOTE     | Layer nn.Dense does not support the setting of channel_wise = true. |

<table>
<thead>
<tr>
<th>Recommended Value</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

---
**quant_enable** specifies whether to quantize a layer. If set to **true**, the layer is to be quantized. If set to **false**, otherwise. If the configuration of a layer is not present, the quantization of the layer is skipped. When the network accuracy is not as expected, locate the quantization-sensitive layers (whose error increases significantly after quantization) in the network, and disable quantization on these layers as needed. There are two methods to identify the quantization sensitive layer. One is based on the model structure. Generally, the accuracy of the first layer, last layer, and layers with a small number of parameters on the network decreases greatly after quantization. In the other method, the accuracy comparison tool can be used to compare the output errors of the original model and the quantized model layer by layer (for example, the cosine similarity is used as a standard, and the similarity must be greater than 0.99). If a layer with a large error is found, the layer is preferentially rolled back.

**Step 6** If the inference accuracy with the model quantized in **Step 5** is as expected, configuration tuning ends. Otherwise, go to **Step 7**.

**Step 7** Tune the values of **activation_quant_params** and **weight_quant_params** in the quantization configuration file.

- Data is clipped to the range \([left, right]\) specified by the **activation_quant_params** parameters. Generally, values distributed near a boundary are sparse, and clip may be performed on all the values, to improve the accuracy. A larger value of **min_percentile** (**max_percentile**) indicates that **left** (**right**) is closer to the minimum value (maximum value) of the to-be-quantized data. **search_range** and **search_step** affect the range of \([left, right]\). Generally, a larger value of **search_range** and a smaller value of **search_step** may achieve higher quantization accuracy, but the quantization takes more time.

- **channel_wise** in **weight_quant_params** determines whether to use a different quantization factor for each channel during weight quantization. If set to **true**, channels are separately quantized using different quantization factors. If set to **false**, all channels are quantized altogether using the same quantization factors. Generally, each channel is independently quantized, and the quantization accuracy is high. Therefore, this mode is recommended. However, the nn.Dense layer is channel-irrelevant. When **channel_wise** is set to **True**, an error message is displayed.

**Step 8** If the quantization accuracy obtained in **Step 7** is as expected, the parameter tuning ends. Otherwise, quantization has a great impact on the accuracy, and quantization cannot be performed, so the quantization configuration needs to be removed.

----End
7.4 Quantization Aware Training
7.4.1 Single-Device Quantization Example

Prerequisites

- **Model**
  For details, see Model.
- **Image dataset**
  Quantization aware training uses a large amount of data to further optimize the quantization parameters. Therefore, the CIFAR-10 dataset is used for quantization aware training. For details about how to download the dataset, see Image Dataset and Calibration Dataset.

Procedure

1. Download the model files.
   For details, see 2.
   Two quantization aware training scripts are provided: one is the resnet50_retrain_sample.py, which allows more flexibility; the other is the encapsulated script run_standalone_train_sample.sh where only a few parameters need to be configured. Choose the method that suits your requirement.
   - Using **resnet50_retrain_sample.py**
     Run the following command in the sample/ResNet50 directory:
     ```bash
     ```
     The command-line options are described as follows.

     **Table 7-20 Quantization aware training command-line options**

<pre><code> | Option   | Description |
 |----------|-------------|
 | -h       | Optional.   |
 |          | Displays help information. |
 | --net    | Optional.   |
 |          | A string.   |
 |          | Default: resnet50 |
 |          | Sets the network. Must be resnet50. |
 | --dataset| Optional.   |
 |          | A string.   |
 |          | Default: cifar10 |
 |          | Sets the image dataset. Must be cifar10. |
</code></pre>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| --run_distribute     | • Optional.  
                        • A bool.  
                        • Default: False  
                        • Runs multi-device quantization aware training. |
| --device_num         | • Optional.  
                        • An int.  
                        • Default: 1  
                        • Sets the number of devices connected to the host. |
| --device_target      | • Optional.  
                        • A string.  
                        • Default: Ascend  
                        • Sets the Ascend backend as the target device. |
| --pre_trained        | • Required.  
                        • A string.  
                        • Default: None  
                        • Sets the checkpoint file directory of the pre-trained ResNet-50. |
| --eval_dataset       | • Required.  
                        • A string.  
                        • Default: None  
                        • Sets the image dataset directory. |
| --train_dataset      | • Required.  
                        • A string.  
                        • Default: None  
                        • Sets the dataset directory for quantization aware training |
| --air AIR            | • Optional.  
                        • A string.  
                        • Default: resnet50_quant_retrain.air  
                        • Sets the file name of the result .air model. |
An example is as follows.

```
python3.7.5 resnet50_retrain_sample.py --train_dataset $HOME/AMCT/dataset/cifar-10-batches-bin --eval_dataset /$HOME/AMCT/dataset/cifar-10-verify-bin --pre_trained $HOME/AMCT/model/resnet50.ckpt
```

Using the `run_standalone_train_sample.sh` encapsulated quantization script

Go to the `sample/ResNet50/scripts` directory and run the following command to start quantization aware training on the ResNet-50 model:

```
sh run_standalone_train_sample.sh [resnet50] [cifar10] [TRAIN_DATASET_PATH] [EVAL_DATASET_PATH] [PRETRAINED_CKPT_PATH]
```

The command-line options are described as follows.

### Table 7-21 Command-line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Optional.</td>
</tr>
<tr>
<td></td>
<td>Displays help information.</td>
</tr>
<tr>
<td>[resnet50]</td>
<td>Required.</td>
</tr>
<tr>
<td></td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td>Default: resnet50</td>
</tr>
<tr>
<td></td>
<td>Sets the model to train. Must be resnet50.</td>
</tr>
<tr>
<td>[cifar10]</td>
<td>Required.</td>
</tr>
<tr>
<td></td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td>Default: cifar10</td>
</tr>
<tr>
<td></td>
<td>Sets the training dataset. Must be cifar10.</td>
</tr>
<tr>
<td>[TRAIN_DATASET_PATH]</td>
<td>Required.</td>
</tr>
<tr>
<td></td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
</tr>
<tr>
<td></td>
<td>Sets the directory of the CIFAR-10 training dataset.</td>
</tr>
</tbody>
</table>
A command example is as follows.

```
sh run_standalone_train_sample.sh resnet50 cifar10 $HOME/AMCT/dataset/cifar-10-batches-bin/$HOME/AMCT/dataset/cifar-10-verify-bin $HOME/AMCT/model/resnet50.ckpt
```

If a message similar to the following is displayed, the quantization aware training is successful:

```
INFO - [AMCT]:[QuantizeTool]: Generate AIR file: xxx/resnet50_quant_retrain.air success
```

3. View the result.

After the quantization aware training is complete, find the following folders and files generated to the `sample/ResNet50` directory:

- `resnet50_quant_retrain.air`: quantized model that can be deployed on the Ascend AI Processor.
- `retrain_quant_config.json`: quantization aware training configuration file that describes how to train each layer in the model. If a quantization aware configuration file already exists in the directory of the quantization aware script, the existing configuration file is overwritten by a new one with the same name in a call to 7.7.3.1 `create_quant_retrain_config`. If not that case, a new configuration file is created.

If the accuracy of model inference drops significantly after quantization, you can modify the `retrain_quant_config.json` file by referring to 7.4.3 `Quantization Configuration`.

- `retrain_result`: quantization aware training result directory.
- `kernel_meta`: directory of operator build outputs.

Calibration is performed as part of the quantization aware training process. After quantization aware training is complete, files such as `config.json` and `amct_dump` are also generated. For details about the files, see 5.

### 7.4.2 Multi-Device Quantization Example

If there are multiple devices in the training environment, you can perform quantization aware training with multiple devices.
Prerequisites

- **Model**
  For details, see Model.

- **Image dataset**
  Quantization aware training uses a large amount of data to further optimize the quantization parameters. Therefore, the CIFAR-10 dataset is used for quantization aware training. For details about how to download the dataset, see Image Dataset and Calibration Dataset.

Procedure

1. Download the model files.
   For details, see 2.

2. Generate the HCCL distributed configuration file.
   The HCCL distributed configuration file depends on the hccn.conf configuration file in the /etc directory on the device in the training environment. Therefore, before generating the HCCL distributed configuration file, ensure that the training environment has been set up and the hccn.conf configuration file exists.

   Go to the ResNet50/src directory and run the following command:

   ```
   python3.7.5 hccl_tools.py --device_num DEVICE_NUM --visible_devices VISIBLE_DEVICES --server_ip SERVER_IP
   ```

   The command-line options are described as follows.

   **Table 7-22 Command-line options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>● Optional.        ● Displays help information.</td>
</tr>
<tr>
<td>--device_num DEVICE_NUM</td>
<td>● Required.         ● A string.</td>
</tr>
<tr>
<td></td>
<td>● Default: &quot;[0,8]&quot;   ● Sets the number of devices used for training. The value range is [0, 8). A maximum of eight devices are supported. The specified device IDs must be consecutive. For example, [0, 4) indicates that devices 0, 1, 2, and 3 are used; [0, 1) indicates that device 0 is used. It is not allowed to specify nonconsecutive device IDs, for example, [3, 6). The value must be enclosed in double quotation marks.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------</td>
</tr>
</tbody>
</table>
| --visible_devices **VISIBLE_DEVICES** | • Optional.  
 • A string.  
 • Default: "0,1,2,3,4,5,6,7"  
 • Uses available devices in sequence. |
| --server_ip **SERVER_IP**   | • Optional.  
 • A string.  
 • Default: **None**  
 • Sets the IP address of the host server. |

A command example is as follows:
```
python3.7.5 hccl_tools.py --device_num "[0,8)"
```

After the preceding command is executed, the `hccl_[device_num]p_[which_device]_[server_ip].json` file is generated in the `src` directory, for example, the `hccl_1p_0_127.0.0.1.json` file in this example. This file records the IP address and device ID of device 0, which are used to determine the device used for subsequent training.

3. Perform distributed quantization aware training.

Go to the `ResNet50/scripts` directory and run the following command:
```
sh run_distribute_train_sample.sh [resnet50] [cifar10] [RANK_TABLE_FILE] [TRAIN_DATASET_PATH] [EVAL_DATASET_PATH] [PRETRAINED_CKPT_PATH]
```

The command-line options are described as follows.

**Table 7-23** Command-line options for distributed quantization aware training

<table>
<thead>
<tr>
<th>Option</th>
<th>Required/Optional</th>
</tr>
</thead>
</table>
| -h         | • Optional.  
 • Displays help information. |
| [resnet50] | • Required.  
 • A string.  
 • Default: **resnet50**  
 • Sets the model to train. Must be **resnet50**. |
| [cifar10]  | • Required.  
 • A string.  
 • Default: **cifar10**  
 • Sets the training dataset. Must be **cifar10**. |
<table>
<thead>
<tr>
<th>Option</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>[RANK_TABLE_FILE]</td>
<td>• Required.</td>
</tr>
<tr>
<td></td>
<td>• A string.</td>
</tr>
<tr>
<td></td>
<td>• Default: None</td>
</tr>
<tr>
<td></td>
<td>• Sets the directory of the HCCL</td>
</tr>
<tr>
<td></td>
<td>configuration file generated in 2.</td>
</tr>
<tr>
<td>[TRAIN_DATASET_PATH]</td>
<td>• Required.</td>
</tr>
<tr>
<td></td>
<td>• A string.</td>
</tr>
<tr>
<td></td>
<td>• Default: None</td>
</tr>
<tr>
<td></td>
<td>• Sets the directory of the CIFAR-10</td>
</tr>
<tr>
<td></td>
<td>training dataset.</td>
</tr>
<tr>
<td>[EVAL_DATASET_PATH]</td>
<td>• Required.</td>
</tr>
<tr>
<td></td>
<td>• A string.</td>
</tr>
<tr>
<td></td>
<td>• Default: None</td>
</tr>
<tr>
<td></td>
<td>• Sets the directory of the CIFAR-10</td>
</tr>
<tr>
<td></td>
<td>test dataset.</td>
</tr>
<tr>
<td>[PRETRAINED_CKPT_PATH]</td>
<td>• Required.</td>
</tr>
<tr>
<td></td>
<td>• A string.</td>
</tr>
<tr>
<td></td>
<td>• Default: None</td>
</tr>
<tr>
<td></td>
<td>• Sets the checkpoint file directory of</td>
</tr>
<tr>
<td></td>
<td>the pre-trained ResNet-50.</td>
</tr>
</tbody>
</table>

A command example is as follows.

```
sh run_distribute_train_sample.sh resnet50 cifar10 ../src/hccl_1p_0_127.0.0.1.json $HOME/AMCT/dataset/cifar-10-batches-bin /$HOME/AMCT/dataset/cifar-10-verify-bin $HOME/AMCT/model/resnet50.ckpt
```

4. View the distributed quantization aware training result.

After the quantization perception aware is complete, the quantization aware training result for each device is generated in the sample/ResNet50/scripts directory, for example, train_parallel0, train_parallel1, train_parallel2, ...

Open the result file of a specific device, for example, train_parallel0.

- `resnet50_quant_retrain.air`: quantized model that can be deployed on the Ascend AI Processor.
- `retrain_quant_config.json`: quantization aware training configuration file that describes how to train each layer in the model. If a quantization aware configuration file already exists in the directory of the quantization aware script, the existing configuration file is overwritten by a new one with the same name in a call to 7.7.3.1 create_quant_retrain_config. If not that case, a new configuration file is created.

If the accuracy of model inference drops significantly after quantization, you can modify the `retrain_quant_config.json` file by referring to 7.4.3 Quantization Configuration.
- **amct_log/amct_mindspore.log**: quantization aware training log file.
- **retrain_result**: quantization aware training result directory.
- **kernel_meta**: directory of operator build outputs.
- **retrain_resultckpt_0**: directory of the checkpoint files saved during training.
- **log**: directory of the distributed training log files, which are redirected from log printing.
- **env.log**: MindSpore log file.

### 7.4.3 Quantization Configuration

#### 7.4.3.1 Overview

If inference based on the **config.json** quantization aware training configuration file generated by the **7.7.3.1 create_quant_retrain_config** call has significant accuracy drop, tune the **config.json** file until the accuracy is as expected. The following is an example of the file content. Keep the layer names unique in the JSON file.

```json
{
  "version":1,
  "conv1":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo":"ulq_quantize"
    },
    "retrain_weight_config":{
      "algo":"arq_retrain",
      "channel_wise":true
    }
  },
  "layer1.0.down_sample_layer.0":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo":"ulq_quantize"
    },
    "retrain_weight_config":{
      "algo":"arq_retrain",
      "channel_wise":true
    }
  },
  "layer1.0.conv1":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo":"ulq_quantize"
    },
    "retrain_weight_config":{
      "algo":"arq_retrain",
      "channel_wise":true
    }
  },
  ...
}
```

#### 7.4.3.2 Configuration File Options

The following describes the configuration options available in the configuration file. Note that Table 7-30 to Table 7-32 are available only when you manually tune the quantization configuration file.
### Table 7-24 version

<table>
<thead>
<tr>
<th>Function</th>
<th>Version number of the quantization configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>int</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Currently, only version 1 is available.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 7-25 retrain_enable

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization aware training enable by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>true or false</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If set to true, quantization aware training is performed on this layer. If set to false, otherwise.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 7-26 retrain_data_config

<table>
<thead>
<tr>
<th>Function</th>
<th>Activation quantization configuration by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>object</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Includes the following parameters:</td>
</tr>
<tr>
<td></td>
<td>• <strong>algo</strong>: selects the quantization algorithm, default to ulq_quantize.</td>
</tr>
<tr>
<td></td>
<td>• <strong>clip_max</strong>: sets the upper limit of the clip quantization algorithm, default to be empty.</td>
</tr>
<tr>
<td></td>
<td>• <strong>clip_min</strong>: sets the lower limit of the clip quantization algorithm, default to be empty.</td>
</tr>
<tr>
<td></td>
<td>• <strong>fixed_min</strong>: fixes the minimum value of the clip quantization algorithm at 0, default to be empty.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>
### Table 7-27: retrain_weight_config

<table>
<thead>
<tr>
<th>Function</th>
<th>Weight quantization configuration by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>Includes the following parameters:</td>
</tr>
<tr>
<td></td>
<td>• <strong>algo</strong>: quantization algorithm select, default to <strong>arq_retrain</strong>.</td>
</tr>
<tr>
<td></td>
<td>• <strong>channel_wise</strong></td>
</tr>
<tr>
<td>Recommended Value</td>
<td>None</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 7-28: algo

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization algorithm by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>• <strong>ulq_quantize</strong>: ULQ clip quantization algorithm.</td>
</tr>
<tr>
<td></td>
<td>• <strong>arq_retrain</strong>: ARQ quantization algorithm.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>Set to <strong>ulq_quantize</strong> for activation quantization or <strong>arq_retrain</strong> for weight quantization.</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 7-29: channel_wise

<table>
<thead>
<tr>
<th>Function</th>
<th>Whether to use different quantization factors for each channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td><strong>true</strong> or <strong>false</strong></td>
</tr>
<tr>
<td>Description</td>
<td>• If set to <strong>true</strong>, channels are separately quantized using different quantization factors.</td>
</tr>
<tr>
<td></td>
<td>• If set to <strong>false</strong>, all channels are quantized altogether using the same quantization factors.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td><strong>true</strong></td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
<tr>
<td><strong>Table 7-30 fixed_min</strong></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td>Lower limit enable of the activation quantization algorithm</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>true or false</td>
</tr>
</tbody>
</table>
| **Description** | ● If set to true, the lower limit of the activation quantization algorithm is fixed at 0.  
● If set to false, the lower limit of the activation quantization algorithm is not fixed.  
If this option is not included, the AMCT automatically sets the lower limit of the activation quantization algorithm according to the graph structure.  
If this option is included: when the upstream layer of the quantization layer is ReLU, you need to manually set this option to true; when the upstream layer of the quantization layer is not ReLU, you need to manually set this option to false. |
| **Recommended Value** | Do not include this option. |
| **Required/Optional** | Optional |

<table>
<thead>
<tr>
<th><strong>Table 7-31 clip_max</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
</tr>
<tr>
<td><strong>Type</strong></td>
</tr>
</tbody>
</table>
| **Value Range** | clip_max>0  
Controls the upper limit max based on the data distribution of the activations at different layers. The recommended value range is as follows:  
[0.3 * max, 1.7 * max] |
| **Description** | If this option is included, the clip upper limit of the activation quantization algorithm is fixed. If this option is not included, the clip upper limit is learned using the IFMR algorithm. |
| **Recommended Value** | Do not include this option. |
| **Required/Optional** | Optional |

<table>
<thead>
<tr>
<th><strong>Table 7-32 clip_min</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>Value Range</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Recommended Value</td>
</tr>
<tr>
<td>Required/Optional</td>
</tr>
</tbody>
</table>

### 7.4.3.3 Configuration Tuning

If the inference accuracy of the model quantized based on the default config.json configuration drops significantly, perform the following steps to tune the quantization aware training configuration file:

1. Execute the quantization script in the amct_mindspore_sample.tar.gz package to perform quantization based on the default configuration generated by the 7.7.3.1 create_quant_retrain_config API. If the quantization accuracy is as expected, the configuration tuning ends. Otherwise, go to 2.

2. Cancel the quantization of certain layers by changing the value of retrain_enable to false. Generally, the input and output layers of a model have a greater impact on the inference result. Therefore, you can try to cancel the quantization of the input and output layers first.

If you have desirable settings for clip_max and clip_min, modify the quantization configuration file as follows.

```json
{
  "version":1,
  "layername1":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo":"ulq_quantize",
      "clip_max":3.0,
      "clip_min":-3.0
    },
    "retrain_weight_config":{
      "algo":"arq_retrain",
      "channel_wise":true
    }
  },
  "layername2":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo":"ulq_quantize",
      "clip_max":3.0,
      "clip_min":-3.0
    },
    "retrain_weight_config":{
      "algo":"arq_retrain",
      "channel_wise":true
    }
  }
}
```
3. Configuration tuning ends if the inference accuracy meets the requirement. Otherwise, it indicates that quantization aware training has severe adverse impact on the inference accuracy. In this case, remove the quantization aware training configuration.

### 7.5 AMCT Update

The latest AMCT release allows you to access to the latest features. Before updating AMCT, uninstall the existing installation by referring to **7.6 AMCT Uninstallation**, and then install the latest version by referring to **7.2 AMCT Installation**.

### 7.6 AMCT Uninstallation

You can uninstall the AMCT as follows:

1. Run the following command in any directory on the Linux server as the AMCT installation user:
   ```bash
   pip3.7.5 uninstall amct_mindspore
   ```
2. When the following information is displayed, enter `y`:
   ```
   Uninstalling amct-mindspore-{version}:
   Would remove:
   ...
   Proceed (y/n)? y
   ```
   If a message similar to the following is displayed, the uninstallation is successful:
   ```
   Successfully uninstalled amct-mindspore-{version}
   ```
   The installed MindSpore will not be uninstalled during the uninstallation.

### 7.7 API Description

#### 7.7.1 Common APIs

##### 7.7.1.1 set_logging_level

**Description**

Sets the logging levels of the log messages printed to the screen and those saved to the `amct_log/amct_mindspore.log` file.

**Prototype**

```python
set_logging_level(print_level='info', save_level='info')
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
</table>
| print_level | Input | Logging level of the log messages printed to the screen.  
- `debug`: DEBUG, INFO, WARNING, and ERROR logs.  
- `info`: INFO, WARNING, and ERROR logs.  
- `warning`: WARNING and ERROR logs.  
- `error`: ERROR logs. | Default: `info`
  
  A string. |
| save_level | Input | Logging level of log messages saved to the `quant_info.log` file.  
- `debug`: DEBUG, INFO, WARNING, and ERROR logs.  
- `info`: INFO, WARNING, and ERROR logs.  
- `warning`: WARNING and ERROR logs.  
- `error`: ERROR logs. | Default: `info`
  
  A string. |

Logging Level

<table>
<thead>
<tr>
<th>Logging Level</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG</td>
<td>Detailed quantization processing information and related debugging information.</td>
</tr>
<tr>
<td>INFO</td>
<td>Brief quantization processing information, including the quantized layer names and BN fusion information.</td>
</tr>
<tr>
<td>WARNING</td>
<td>Warning messages during quantization.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error messages during quantization.</td>
</tr>
</tbody>
</table>

The logging level is case insensitive. That is, `Info`, `info`, and `INFO` are all valid values.

Returns

None

Example

```python
import amct_mindspore as amct
amct.set_logging_level(print_level="info", save_level="info")
```

7.7.2 Post-training Quantization
7.7.2.1 create_quant_config

Description

Applies to post-training quantization. Finds all quantization-capable layers in a graph, creates a quantization configuration file, and writes the quantization configuration information of the quantization-capable layers to the configuration file.

Restrictions

Due to data type conversion, the quantization factors in the generated quantization configuration file might be different from those in the simplified configuration file. However, the accuracy is not affected.

Prototype

create_quant_config(config_file, network, *input_data, skip_layers=None, activation_offset=True, config_defination=None)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the quantization configuration file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The existing file (if available) in the directory will be overwritten upon this API call.</td>
<td></td>
</tr>
<tr>
<td>input_data</td>
<td>Input</td>
<td>User network input data (The format and shape must be correct. The data can be randomly generated.)</td>
<td>An object that can be converted into a MindSpore Tensor, for example, a numpy.ndarray object. This parameter is a variable parameter and applies to the scenario where the user network has multiple inputs.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
<td>-----------------------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| skip_layers     | Input        | Quantization-capable layers to skip.    | Default: **None**  
A list of strings.  
Restrictions:  
- Keyword parameter, required when the **input_data** variable parameter is used.  
- If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect. |
| activation_offset | Input      | Whether to quantize activations with offset. | Default: **true**  
A bool.  
Restrictions:  
- Keyword parameter, required when the **input_data** variable parameter is used.  
- If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect. |
### Returns

None

### Outputs

Outputs a quantization configuration file in JSON format. (When quantization is performed again, the existing one output by this API call will be overwritten.)

```json
{
    "version":1,
    "activation_offset":true,
    "do_fusion":true,
    "skip_fusion_layers":[]
}
```

```
"conv1":{
    "quant_enable":true,
    "activation_quant_params":{
        "max_percentile":0.999999,
        "min_percentile":0.999999,
        "search_range":{
            "0.7",
            "1.3"
        },
        "search_step":0.01
    },
    "weight_quant_params":{
        "channel_wise":true
    }
}
```

```
"end_point":{
    "quant_enable":true,
    "activation_quant_params":{
        "max_percentile":0.999999,
        "min_percentile":0.999999,
```
"search_range":[
    0.7,
    1.3
],
"search_step":0.01
},
"weight_quant_params":{
    "channel_wise":false
}
},
"layer1.0.conv1":{
    "quant_enable":true,
    "activation_quant_params":{
        "max_percentile":0.999999,
        "min_percentile":0.999999,
        "search_range":[
            0.7,
            1.3
        ],
        "search_step":0.01
    },
    "weight_quant_params":{
        "channel_wise":true
    }
},
"layer1.0.conv2":{
    "quant_enable":true,
    "activation_quant_params":{
        "max_percentile":0.999999,
        "min_percentile":0.999999,
        "search_range":[
            0.7,
            1.3
        ],
        "search_step":0.01
    },
    "weight_quant_params":{
        "channel_wise":true
    }
}
}

Example

```python
from amct_mindspore import create_quant_config
# Create a quantization configuration file from arguments.
input_data = np.random.uniform(0.0, 1.0, size=[32, 1, 32, 32]).astype(np.float32)
create_quant_config("./configs/config.json",
    network,
    input_data,
    skip_layers=None,
    activation_offset=True)
# Generate a quantization configuration file from the simplified configuration file.
create_quant_config("./configs/config.json",
    network,
    input_data,
    config_definition="./configs/quant.cfg")
```

7.7.2.2 quantize_model

Description

Applies to post-training quantization. Quantizes a graph based on the quantization configuration file `config_file`, inserts the activation quantization layers, and returns the modified network.
Prototype

```
network = quantize_model(config_file, network, *input_data)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input / Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization configuration file generated by the user, which is used to specify the configuration of the quantization layer in the network.</td>
<td>A string.</td>
</tr>
<tr>
<td>input_data</td>
<td>Input</td>
<td>User network input data (The format and shape must be correct. The data can be randomly generated.)</td>
<td>An object that can be converted into a MindSpore Tensor, for example, a <code>numpy.ndarray</code> object. This parameter is variable and supports multi-input networks.</td>
</tr>
</tbody>
</table>


Returns

The modified network, whose quantization factors need to be obtained via network inference.

Example

```
import amct_mindspore as amct
# Create a network to be quantized.
network = build_network()
# Generate random input data. The format and type of the data must be the same as those of the network input.
input_data = np.random.uniform(0.0, 1.0, size=[32, 1, 32, 32]).astype(np.float32)
# Quantize the model.
calibration_network = amct.quantize_model(config_file=’./configs/config.json’, network, input_data)
# Perform single-batch inference with the modified model.
model =Model(calibration_network, loss_fn=loss, metrics={'top_1_accuracy', 'top_5_accuracy'})
model.eval(dataset, dataset_sink_mode=False)
```
7.7.2.3 save_model

Description

Applies to post-training quantization. Saves the model returned by the 7.7.2.2 quantize_model API call in AIR format. The model needs to be inferred in one batch to obtain the correct quantization factors. If the model is directly saved to AIR format without inference, the saved quantization factors are incorrect.

Prototype

```
save_model(file_name, network, *input_data)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>Input</td>
<td>Directory (including the filename) of the result AIR file.</td>
<td>A string.</td>
</tr>
<tr>
<td>network</td>
<td>Input</td>
<td>Quantized model generated from 7.7.2.2 quantize_model.</td>
<td>A MindSpore Cell object.</td>
</tr>
<tr>
<td>input_data</td>
<td>Input</td>
<td>User network input data (The format and shape must be correct. The data can be randomly generated.)</td>
<td>An object that can be converted into a MindSpore Tensor, for example, a numpy.ndarray object. This parameter is variable and supports multi-input networks.</td>
</tr>
</tbody>
</table>

Returns

None

Outputs

Outputs an AIR model file for inference on the Ascend AI Processor.

When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.

Example

```
import amct_mindspore as amct
input_data = np.random.uniform(0.0, 1.0, size=[32, 1, 32, 32]).astype(np.float32)
amct.save_model('lenet_geir', calibration_network, input_data)
```
7.7.3 Quantization Aware Training

7.7.3.1 create_quant_retrain_config

**Description**
Applies to quantization aware training. Finds all quantization-capable layers in a graph, creates a quantization configuration file, and writes the quantization configuration information of the quantization-capable layers to the configuration file.

**Restrictions**
None

**Prototype**

```python
create_quant_retrain_config(config_file, network, *input_data, config_definition=None)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the configuration file, including the file name. The existing file (if available) in the directory will be overwritten upon this API call.</td>
<td>A string.</td>
</tr>
<tr>
<td>input_data</td>
<td>Input</td>
<td>User network input data (the format and shape of the data must be correct, and the data can be randomly generated) for graph building.</td>
<td>An object that can be converted into a MindSpore Tensor, for example, a numpy.ndarray object. This parameter is variable and supports multi-input networks.</td>
</tr>
<tr>
<td>Paramet er</td>
<td>Input/Retu rn</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>config_def ination</td>
<td>Input</td>
<td>Simplified quantization configuration file <code>quant.cfg</code>, which is generated from the <code>retrain_config_mindspore.proto</code> file. The <code>retrain_config_mindspore.proto</code> file is stored in <code>/amct_mindspore/proto/retrain_config_mindspore.proto</code> in the AMCT installation path. For details about the parameters in the <code>retrain_config_mindspore.proto</code> template and the generated simplified quantization configuration file <code>quant.cfg</code>, see 7.8.2 Simplified Quantization Aware Training Configuration File.</td>
<td>Default: None</td>
</tr>
</tbody>
</table>

### Returns
None

### Outputs
Outputs a quantization aware training configuration file in JSON format. The existing configuration file (if available) in the directory will be overwritten upon this API call. An example is as follows.

```json
{
  "version":1,
  "conv":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo"="ulq_quantize"
    },
    "retrain_weight_config":{
      "algo"="arq_retrain",
      "channel_wise":true
    }
  },
  "layer1.0.down_sample_layer.0":{
    "retrain_enable":true,
    "retrain_data_config":{
      "algo"="ulq_quantize"
    },
    "retrain_weight_config":{
      "algo"="arq_retrain",
      "channel_wise":true
    }
  },
  "layer1.0.conv1":{
    "retrain_enable":true,
    "retrain_data_config":{
```
"algo":"ulq_quantize",
"retrain_weight_config":{
  "algo":"arq_retrain",
  "channel_wise":true
}
},
...

Example

import amct_mindspore as amct
import numpy as np

network = resnet(10)
network.set_train(True)
ckpt_path = os.path.join(CUR_DIR, './ckpt/resnet50.ckpt')
param_dict = load_checkpoint(ckpt_path)
load_param_into_net(network, param_dict)

input_data = np.random.uniform(0.0, 1.0, size=[32, 3, 224, 224]).astype(np.float32)
config_file = os.path.join(CUR_DIR, '/retrain_quant_config.json')
amct.create_quant_retrain_config(config_file, network, input_data)

7.7.3.2 UlqInitializer

Description

Applies to quantization aware training. Constructs an initializer for initializing operator parameters related to quantization aware training as an argument to the 7.7.3.3 create_quant_retrain_model call.

Restrictions

None

Prototype

initializer = UlqInitializer(scale_offset_record)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale_offset_record</td>
<td>Input</td>
<td>Quantization factor record file generated from calibration.</td>
<td>None</td>
</tr>
<tr>
<td>initializer</td>
<td>Return</td>
<td>An initializer for initializing operator parameters for quantization-aware training. Initializer instance.</td>
<td>None</td>
</tr>
</tbody>
</table>
Returns

Initializer instance.

Outputs

None

Example

scale_offset_record_file = ['./amct_dump/calibration_record.txt']
initializer = UlqInitializer(scale_offset_record_file)

7.7.3.3 create_quant_retrain_model

Description

Applies to quantization aware training. Looks up the network layers to be quantized based on the quantization configuration, inserts quantization-aware layers, outputs the retrained network for further training.

Restrictions

Before this API is called, the pre-trained checkpoint file has been loaded to the original network.

Prototype

retrain_network = create_quant_retrain_model(config_file, network, initializer, *input_data)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input / Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the configuration file generated by the user.</td>
<td>A string.</td>
</tr>
<tr>
<td><strong>Parameter</strong></td>
<td><strong>Input / Return</strong></td>
<td><strong>Description</strong></td>
<td><strong>Restriction</strong></td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>initializer</td>
<td>Input</td>
<td><strong>Initializer</strong> instance for initializing the ULQ layer. Perform initialization in advance. Pass the scale_offset record file obtained by calibration to construct an <strong>Initializer</strong> instance. The value <strong>None</strong> is supported, indicating that the default initial values are used to initialize <strong>clip_min</strong> and <strong>clip_max</strong>. However, this value is not recommended because the result is not the optimal.</td>
<td>An <strong>Initializer</strong> instance Default: <strong>None</strong></td>
</tr>
<tr>
<td>input_data</td>
<td>Input</td>
<td>User network input data (the format and shape of the data must be correct, and the data can be randomly generated) for graph building.</td>
<td>An object that can be converted into a MindSpore Tensor, for example, a <strong>numpy.ndarray</strong> object. This parameter is variable and supports multi-input networks.</td>
</tr>
</tbody>
</table>

**Returns**

Network for quantization aware training.

**Outputs**

None

**Example**

```python
import amct_mindspore as amct
from amct_mindsporeinitializer import UlqInitializer
import numpy as np

network = resnet(10)
network.set_train(True)
ckpt_path = os.path.join(CUR_DIR, './ckpt/resnet50.ckpt')
param_dict = load_checkpoint(ckpt_path)
load_param_into_net(network, param_dict)

input_data = np.random.uniform(0.0, 1.0, size=[32, 3, 224, 224]).astype(np.float32)
config_file = os.path.join(CUR_DIR, './retrain_quant_config.json')
```
7.7.3.4 restore_quant_retrain_model

Description

Applies to quantization aware training. Loads the checkpoint file and outputs the modified retrained network for further quantization aware training. This API is used to modify the network based on the checkpoint file and quantization configuration file saved during the quantization aware training when the training is interrupted.

Restrictions

The checkpoint file is saved in the process of quantization aware training.

Prototype

```
retrain_network = restore_quant_retrain_model(config_file, network, checkpoint_path, *input_data)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the configuration file generated by the user.</td>
<td>A string. The <code>config.json</code> file passed to this call must be the same as that pass to the <code>create_quant_retrain_model</code> call.</td>
</tr>
<tr>
<td>checkpoint_path</td>
<td>Input</td>
<td>Directory of the checkpoint file saved during quantization aware training.</td>
<td>A string.</td>
</tr>
</tbody>
</table>
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_data</td>
<td>Input</td>
<td>User network input data (the format and shape of the data must be correct, and the data can be randomly generated) for graph building.</td>
<td>An object that can be converted into a MindSpore Tensor, for example, a <code>numpy.ndarray</code> object. This parameter is variable and supports multi-input networks.</td>
</tr>
</tbody>
</table>

### Returns

Network for quantization aware training.

### Outputs

None

### Example

```python
import amct_mindspore as amct
from amct_mindspore.initializer import UlqInitializer
import numpy as np

network = resnet(10)
network.set_train(True)
ckpt_path = os.path.join(CUR_DIR, './ckpt/resnet50.ckpt')
param_dict = load_checkpoint(ckpt_path)
load_param_into_net(network, param_dict)
input_data = np.random.uniform(0.0, 1.0, size=[32, 3, 224, 224]).astype(np.float32)
config_file = os.path.join(CUR_DIR, './retrain_quant_config.json')

# 1. create the quant config
amct.create_quant_retrain_config(config_file, network, input_data)

# 2. get the quantization aware training retrain_network
retrain_network = amct.create_quant_retrain_model(config_file, network, initializer, input_data)

# 3. train the retrain_network and save checkpoint file
......

# 4. restore the retrain_model from a checkpoint and keep on training
checkpoint_path = './resnet50_1-1562.ckpt'
retrain_network = amct.restore_quant_retrain_model(config_file, network, checkpoint_path, input_data)
```
7.7.3.5 `save_quant_retrain_model`

**Description**

Applies to quantization aware training. Saves a retrained model to a deployable model file.

**Restrictions**

None

**Prototype**

```
save_quant_retrain_model(config_file, file_name, network, *input_data)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the configuration file generated by the user.</td>
<td>A string.</td>
</tr>
<tr>
<td>file_name</td>
<td>Input</td>
<td>AIR model save path.</td>
<td>A string.</td>
</tr>
<tr>
<td>network</td>
<td>Input</td>
<td>Retrained network generated by 7.7.3.3 <code>create_quant_retrain_model</code> or 7.7.3.4 <code>restore_quant_retrain_model</code>.</td>
<td>A MindSpore Cell object.</td>
</tr>
<tr>
<td>input_data</td>
<td>Input</td>
<td>User network input data (the format and shape of the data must be correct, and the data can be randomly generated) for graph building.</td>
<td>An object that can be converted into a MindSpore Tensor, for example, a <code>numpy.ndarray</code> object. This parameter is variable and supports multi-input networks.</td>
</tr>
</tbody>
</table>

**Returns**

None

**Outputs**

Outputs an AIR model file for inference on the Ascend AI Processor.
When quantization aware training is performed again, the existing files in the output directory will be overwritten upon this API call.

**Example**

```python
import amct_mindspore as amct
from amct_mindspore.initializer import UlqInitializer
import numpy as np

network = resnet(10)
network.set_train(True)
ckpt_path = os.path.join(CUR_DIR, './ckpt/resnet50.ckpt')
param_dict = load_checkpoint(ckpt_path)
load_param_into_net(network, param_dict)

input_data = np.random.uniform(0.0, 1.0, size=[32, 3, 224, 224]).astype(np.float32)
config_file = os.path.join(CUR_DIR, './retrain_quant_config.json')

# 1. create the quant config
amct.create_quant_retrain_config(config_file, network, input_data)

# 2. get the quant aware training retrain_network
retrain_network = amct.create_quant_retrain_model(config_file, network, initializer, input_data)

# 3. train the modified retrain_network
...

# 4. export the retrain_network to deploy model
amct.save_quant_retrain_model(config_file, './resnet50_deploy', retrain_network, input_data)
```

7.8 Appendixes

7.8.1 Simplified Post-training Quantization Configuration File

Table 7-33 describes the parameters in the `calibration_config_mindspore.proto` template.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified post-training quantization configuration file of AMCT.</td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>activation_offset</td>
<td></td>
<td>Whether to quantize activations with offset.</td>
</tr>
<tr>
<td>Repeated</td>
<td>string</td>
<td>skip_layers</td>
<td></td>
<td>Layers to skip during quantization.</td>
</tr>
<tr>
<td>Repeated</td>
<td>string</td>
<td>skip_layer_types</td>
<td></td>
<td>Types of layers to skip during quantization.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------</td>
<td>-----------------------</td>
<td>--------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Optional</td>
<td>Calibration Config</td>
<td>common_config</td>
<td></td>
<td>Common quantization configuration. If a layer is not overridden by <code>override_layer_types</code> or <code>override_layer_configs</code>, this configuration is used.</td>
</tr>
<tr>
<td>Repeated</td>
<td>OverrideLayerType</td>
<td>override_layer_types</td>
<td></td>
<td>Type of layers to override.</td>
</tr>
<tr>
<td>Repeated</td>
<td>OverrideLayerType</td>
<td>override_layer_configs</td>
<td></td>
<td>Layer to override.</td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>do_fusion</td>
<td></td>
<td>BN fusion switch. Defaults to <code>true</code>.</td>
</tr>
<tr>
<td>Repeated</td>
<td>string</td>
<td>skip_fusion_layers</td>
<td></td>
<td>Layers to skip in BN fusion.</td>
</tr>
<tr>
<td>OverrideLayerType</td>
<td>string</td>
<td>layer_type</td>
<td></td>
<td>Type of layers to quantize.</td>
</tr>
<tr>
<td>Required</td>
<td>Calibration Config</td>
<td>calibration_config</td>
<td></td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td>OverrideLayer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization configuration overriding by layer.</td>
</tr>
<tr>
<td>Required</td>
<td>string</td>
<td>layer_name</td>
<td></td>
<td>Layer to override.</td>
</tr>
<tr>
<td>Required</td>
<td>Calibration Config</td>
<td>calibration_config</td>
<td></td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td>Calibration Config</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Calibration-based quantization configuration.</td>
</tr>
<tr>
<td>-</td>
<td>ARQuantize</td>
<td>arq_quantize</td>
<td></td>
<td>Weight quantization algorithm. <strong>arq_quantize</strong>: ARQ algorithm configuration.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
<td>--------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-</td>
<td>Optional</td>
<td>FMRQuantize</td>
<td>ifmr_quantize</td>
<td>Activation quantization algorithm. <strong>ifmr_quantize</strong>: IFMR algorithm configuration.</td>
</tr>
<tr>
<td>ARQuantize</td>
<td>Optional</td>
<td>-</td>
<td>-</td>
<td>ARQ quantization algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>channel_wise</td>
<td>Whether to use different quantization factors for each channel.</td>
</tr>
<tr>
<td>FMRQuantize</td>
<td>Optional</td>
<td>-</td>
<td>-</td>
<td>FMR quantization algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>search_range_start</td>
<td>Quantization factor search start.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>search_range_end</td>
<td>Quantization factor search end.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>search_step</td>
<td>Quantization factor search step.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>max_percentile</td>
<td>Upper search limit.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>min_percentile</td>
<td>Lower search limit.</td>
</tr>
</tbody>
</table>

The following is an example simplified quantization configuration file (**quant.cfg**).

```yaml
# Global quantization parameters
activation_offset : true
skip_layers : "conv_1"
skip_layer_types:"nn.Conv2d"
do_fusion: true
skip_fusion_layers : "conv_1"
common_config : {
  arq_quantize : {
    channel_wise : true
  }
  ifmr_quantize : {
    search_range_start : 0.7
    search_range_end : 1.3
    search_step : 0.01
    max_percentile : 0.999999
    min_percentile : 0.999999
  }
}
```
### 7.8.2 Simplified Quantization Aware Training Configuration File

Table 7-34 describes the parameters in the `retrain_config_mindspore.proto` template.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTRetrainConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified quantization aware training configuration file of AMCT.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>RetrainDataQuantConfig</td>
<td>retrain_data_quant_config</td>
<td>Activation retrain configuration.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>RetrainWeightQuantConfig</td>
<td>retrain_weight_quant_config</td>
<td>Weight retrain configuration.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_layers</td>
<td>Layers to skip during quantization.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>RetrainOverrideLayer</td>
<td>override_layer_configs</td>
<td>Layer to override.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_layer_types</td>
<td>Types of layers to skip during quantization.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>RetrainOverrideLayerType</td>
<td>override_layer_types</td>
<td>Types of layers to override.</td>
</tr>
<tr>
<td>RetrainDataQuantConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Activation retrain configuration.</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>ULQuantize</td>
<td>ulq_quantize</td>
<td>Activation quantization algorithm. Only ULQ is supported in the current version.</td>
</tr>
<tr>
<td>ULQuantize</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ULQ algorithm configuration.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td>-----------</td>
<td>--------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ClipMaxMin</td>
<td>Optional</td>
<td>ClipMaxMin</td>
<td>clip_max_min</td>
<td>Initial upper and lower limits. If it is not specified, IFMR is used for initialization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>fixed_min</td>
<td>Whether to fix the lower limit at 0. Set to true for ReLU or false for other algorithms.</td>
</tr>
<tr>
<td>RetrainWeightQuantConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Weight retrain configuration.</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>ARQRetrain</td>
<td>arq_retrain</td>
<td>Weight quantization algorithm. Only ARQ is supported in the current version.</td>
</tr>
<tr>
<td>ARQRetrain</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ARQ algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>channel_wise</td>
<td>Channel-wise ARQ enable.</td>
</tr>
<tr>
<td>RetrainOverrideLayer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Layer overriding configuration.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>layer_name</td>
<td>Layer name.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>RetrainData QuantConfig</td>
<td>retrain_data_quant_config</td>
<td>Activation quantization configuration to apply.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>RetrainWeightQuantConfig</td>
<td>retrain_weight_quant_config</td>
<td>Weight quantization configuration to apply.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------</td>
<td>-------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RetrainOverrideLayerType</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Types of layers to override.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>layer_type</td>
<td>Layer type.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>RetrainDataQuantConfig</td>
<td>retrain_data_quant_config</td>
<td>Activation quantization configuration to apply.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>RetrainWeightQuantConfig</td>
<td>retrain_weight_quant_config</td>
<td>Weight quantization configuration to apply.</td>
</tr>
</tbody>
</table>

The following is an example simplified quantization aware training configuration file (`quant.cfg`).

```yaml
# Global quantization parameters
retrain_data_quant_config:
  ulq_quantize:
    clip_max_min:
      clip_max: 6.0
      clip_min: -6.0
    fixed_min: true

skip_layers: "conv2"
skip_layer_types: "nn.Conv2D"

override_layer_types:
  layer_type: "nn.Conv2D"
  retrain_weight_quant_config:
    arq_retrain:
      channel_wise: false

override_layer_configs:
  layer_name: "conv2"
  retrain_data_quant_config:
    ulq_quantize:
      clip_max_min:
        clip_max: 3.0
        clip_min: -3.0
      fixed_min: true

  retrain_weight_quant_config:
    arq_retrain:
      channel_wise: true
```

CANN
Development Auxiliary Tool Guide (Training)

7 AMCT Instructions (MindSpore)

Issue 01 (2021-07-01) Copyright © Huawei Technologies Co., Ltd. 378
### 7.8.3 Quantization Factor Record File

**Prototype**

The quantization factor record file is a serialized data structure file based on Protobuf. You can generate a quantized model file by using the quantization configuration file, original network model file, and the quantization factor record file. The Protobuf prototype is defined as follows (find the code in the `/amct_mindspore/proto/scale_offset_record_mindspore.proto` file in the AMCT installation directory).

```protobuf
syntax = "proto2";
package AMCTMindspore;

message SingleLayerRecord {
  optional float scale_d = 1;
  optional int32 offset_d = 2;
  repeated float scale_w = 3;
  repeated int32 offset_w = 4;
  repeated uint32 shift_bit = 5;
  optional bool skip_fusion = 9 [default = false];
}

message MapFiledEntry {
  optional string key = 1;
  optional SingleLayerRecord value = 2;
}

message ScaleOffsetRecord {
  repeated MapFiledEntry record = 1;
}
```

The parameters are described as follows.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleLayerRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization factors for quantization.</td>
</tr>
<tr>
<td>Optional</td>
<td>float</td>
<td>scale_d</td>
<td></td>
<td>Scale factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td>Optional</td>
<td>int32</td>
<td>offset_d</td>
<td></td>
<td>Offset factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td>Message</td>
<td>Required / Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------</td>
<td>--------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Repetitive float scale_w</td>
<td>Scale factor for weight quantization. Scalar mode (quantizing the weight of the current layer in a unified manner) and vector mode (quantizing the weight of the current layer in channel-wise mode) are supported. Only nn.Conv2d dilation and nn.DepthwiseConv2d layers support channel-wise quantization.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Repetitive int32 offset_w</td>
<td>Offset factor for weight quantization. Similar to scale_w, it also supports the scalar and vector quantization modes and the dimension configuration must be the same as that of scale_w. Currently weight quantization with offset is not supported, and offset_w must be 0.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Repetitive uint32 shift_bit</td>
<td>Shift factor. Reserved.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional bool skip_fusion</td>
<td>Whether to skip Conv+BN fusion at the current layer. Defaults to false, indicating that the preceding fusion type is performed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScaleOffsetRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Map structure. To ensure compatibility, the discrete map structure is used.</td>
</tr>
</tbody>
</table>
| Repetitive MapFiledEntry record | Each records a quantization factor of a quantization layer and consists of two members:  
  - **key**: layer name.  
  - **value**: quantization factor defined by SingleLayerRecord. |
| Optional string key | Layer name. |
| Optional SingleLayerRecord value | Quantization factor configuration. |
Beware that the Protobuf protocol does not report an error for repeated settings of optional fields. Instead, the most recent settings are used.

For general quantization layers, a range of parameters need to be configured, including scale_d, offset_d, scale_w, and offset_w.

```python
record {
  key: "conv1"
  value {
    scale_d: 0.020284997
    offset_d: -8
    scale_w: 0.00039157062
    offset_w: 0
  }
}
record {
  key: "layer1.0.conv1"
  value {
    scale_d: 0.050864004
    offset_d: -128
    scale_w: 0.0023167059
    offset_w: 0
  }
}
record {
  key: "layer1.0.conv2"
  value {
    scale_d: 0.026584487
    offset_d: -128
    scale_w: 0.0010260359
    offset_w: 0
  }
}
```

**Quantization Factors**

The scale (a floating-point number) and offset quantization factors need to be provided for data and weight quantization. AMCT uses a unified quantization data structure. See the following expression.

\[
data_{\text{int8}} = \text{clip}_{\text{int8}} \left( \text{round} \left( \frac{\text{data}_{\text{float}}}{\text{scale}} \right) + \text{offset} \right)
\]

The value ranges are as follows:

- \( \text{scale} \in \left[ \text{FLT\_EPSILON}, \frac{1}{\text{FLT\_EPSILON}} \right] \), \( \text{FLT\_EPSILON} \approx 1.1920929 \times 10^{-7} \)
- \( \text{offset} \in [-128, 127] \)

Quantization algorithms are classified into symmetric quantization algorithms and asymmetric quantization algorithms.

1. Symmetric quantization algorithm:

   The original high-precision data and quantized int8 data are converted into \( \text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int8}} \), where \text{scale} \ is a float32. To indicate positive and negative numbers, the signed int8 data type is used for \( \text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int8}} \). The following describes how to convert the original data into the int8 format. \text{round} \ is a rounding function. The value to be determined by the quantization algorithm is the constant \text{scale}. 
Quantization of the weights and activations may be summarized as a process of searching for a scale. Because \( \text{data}_{\text{int8}} \) is a signed number, to ensure symmetry of the ranges represented by positive and negative values, an absolute value operation is first performed on all data, so that the range of the to-be-quantized data is changed to \( \text{data}_{\text{int8}} \), and then \( \text{scale} \) is determined. The range of positive int8 values is \([0, 127]\). Therefore, \( \text{scale} \) can be computed as follows:

\[
\text{scale} = \frac{\text{data}_{\text{max}}}{127}
\]

Therefore, the range of the int8 values is \([-128 \times \text{scale}, 127 \times \text{scale}]\). Data beyond the range \([-128 \times \text{scale}, 127 \times \text{scale}]\) is saturated to a boundary value, and then the quantization operation shown in the formula is performed.

2. Asymmetric quantization algorithm:

The difference from symmetric quantization algorithms lies in the data conversion mode. The \( \text{scale} \) and \( \text{offset} \) constants also need to be determined.

\[
\text{data}_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset})
\]

The uint8 data is converted through calculation based on the original high-accuracy data, which is shown in the following formula:

\[
\text{data}_{\text{uint8}} = \text{round}\left(\frac{\text{data}_{\text{float}}}{\text{scale}} - \text{offset}\right)
\]

\( \text{scale} \) is an fp32, \( \text{data}_{\text{uint8}} \) is an unsigned int8, and \( \text{offset} \) is an int8. The data range is \([\text{scale} \times \text{offset}, \text{scale} \times (255 + \text{offset})]\). If a value range of the to-be-quantized data is \([\text{data}_{\text{min}}, \text{data}_{\text{max}}]\), \( \text{scale} \) and \( \text{offset} \) are computed as follows:

\[
\text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}, \quad \text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}
\]

Unified quantization data format: By performing simple data conversion of the asymmetric quantization algorithm formula, the quantized data and the symmetric quantization algorithm are of the same type, that is, int. The specific conversion process is as follows:

int8 quantization is used as an example for description. The input original floating-point data is \( \text{data}_{\text{float}} \), the original quantized fixed-point number is \( \text{data}_{\text{float}} \), the quantization scale is \( \text{scale} \), and the quantization offset is \( \text{data}_{\text{float}} \) (the algorithm requires zero crossing to avoid too much accuracy drop). The calculation principle of quantization is as follows:

\[
\text{data}_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset}) = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset} + 128) = \text{scale} \times (\text{data}_{\text{int8}} - \text{offset})
\]

Where,

\[
\text{data}_{\text{int8}} = \text{data}_{\text{uint8}} - 128 \in [-128, 127], \quad \text{offset}' = -(\text{offset} + 128) \in [-128, 127]
\]

Through the foregoing conversion, the data may also be converted into the
int8 format. After \texttt{scale} and the converted \texttt{offset} are determined, the int8 data converted from the original floating-point data is as follows:

\[
data_{\text{int8}} = \text{clip}\left(\text{round}\left(\frac{\text{data}_{\text{float}}}{\text{scale}}\right) + \text{offset}'\right)\]

7.9 FAQs

7.9.1 What Do I Do If a Build Error Occurs in Quantization Aware Training?

\textbf{Symptom}

In the mixed-precision training scenario, if the quantization script has been executed properly during quantization aware training, the following error does not affect the service functionality:

```
2020-11-17 12:01:34.978 INFO : [AMCT][Detacher]: Do <client_mindspore_optimizer_delete_get_layers_pass/DeleteGetLayersPass-> [ERROR] PIPELINE[7622 python3]:2020-11-17 12:01:35.133.063 [mindspore/ccsrc/pipelin...541 [Compile]
```

\textbf{Possible Cause}

When saving the result model, AMCT builds the original graph. If the build fails, AMCT will convert the original graph to FP32 to rebuild the graph.

\textbf{Solution}

You can safely ignore the error messages, which do not affect the service functionality.

7.9.2 What Do I Do If a TypeError Occurs in Quantization Aware Training?

\textbf{Symptom}

In the mixed-precision training scenario, after quantization aware training is performed and the 7.7.3.3 \texttt{create_quant_retrain_model} API is called, the script throws an TypeError, as shown in the following figure.
Possible Cause

After 7.7.3.3 create_quant_retrain_model is called, nn.Dense objects on the original network are replaced with DenseQatBlock objects, and nn.Conv2D objects are replaced with Conv2dQatBlock objects. Before executing the quantization aware training script, you need to import the preceding two objects.

Solution

Import the preceding two objects to your quantization aware training script.

Assume that the original script is as follows.

```python
def switch_precision(net, data_type, config):
    if config.platform == "Ascend":
        net.to_float(data_type)
    for _, cell in net.cells_and_names():
        if isinstance(cell, (nn.Dense, )):
            cell.to_float(mstype.float32)
```

Tweak the script as follows.

```python
from amct_mindspore.cells.dense_qat_cell import DenseQatBlock
from amct_mindspore.cells.conv2d_qat_cell import Conv2dQatBlock
def switch_precision(net, data_type, config):
    if config.platform == "Ascend":
        net.to_float(data_type)
    for _, cell in net.cells_and_names():
        if isinstance(cell, (nn.Dense, DenseQatBlock)):
            cell.to_float(mstype.float32)
```
8 AMCT Instructions (AscendCL)

8.1 Introduction

8.1.1 Overview

This document describes how to quantize a model using Ascend Model Compression Toolkit (AMCT) by using the AscendCL API. In the quantization process, the precision of model weights and activations is reduced to make the model more compact, improving the compute efficiency while lowering the transfer latency.

Currently, AMCT is able to quantize a Caffe or TensorFlow (.pb) model into an .air model. The .air model can be built by Ascend Tensor Compiler (ATC) into an offline model that can run inference on Ascend 310 AI Processor.

**Figure 8-1** shows the AMCT architecture.
1. Install the development environment on the Ascend 910 device, install AMCT in the environment, and perform quantization to generate a quantized model (.air).

2. Copy the quantized model (.air) to the Ascend 310 environment (you are advised to set up the development and operating environments on the same server). In the Ascend 310 environment, use ATC to build the .air model into an .om offline model adapted to the Ascend AI Processor.

3. Run inference on the .om offline model in the Ascend 310 environment.

**8.1.2 Features**

There are two forms of quantization: post-training quantization and quantization aware training. Currently, only post-training quantization is supported.

Post-training quantization refers to quantizing the weights of an already-trained model from float32 to int8 and calibrating and quantizing the activations by using a small calibration dataset. For details about the quantization workflow, see 8.3 Quantization. (It is not supported to run a post-training quantization on more than one NPU.) Post-training quantization is classified into weight quantization and activation quantization according to different quantization objects, as described in the following:

- **Calibration dataset**
  During the calibration process, the algorithm uses each piece of data in the calibration dataset as input, accumulates the input data of each layer (or operator) that needs to be quantized, and uses the accumulated data as input data of the quantization algorithm to determine the quantization factors. Because the quantization factors and the accuracy of the quantized model are closely related to the selection of the calibration dataset, you are advised to calibrate the model using a subset of images from the validation dataset.

- **Activation quantization**
  Activation quantization is to collect statistics on the input data of each layer (or operator) to be quantized, to find the optimal pair of scale and offset per layer (or operator). For details, see 8.7.3 Quantization Factor Record File. Activations are the intermediate compute results of model inference. The value ranges are input-specific. Therefore, a group of reference inputs (a
calibration dataset) needs to be used as incentives to record the input data of each layer (or operator) to be quantized for searching for quantization factors (scale and offset).

- **Weight quantization**
  After model training, the weights and the value ranges are determined. Therefore, quantization may be directly performed based on the value range of each weight.

The layers that support quantization are listed as follows.

**Table 8-1** Layers supporting quantization and the corresponding layers defined in Ascend IRs

<table>
<thead>
<tr>
<th>Framework</th>
<th>Layer</th>
<th>Restriction</th>
<th>Ascend IR-defined Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td>InnerProduct</td>
<td>transpose = false, axis = 1</td>
<td>FullyConnect</td>
</tr>
<tr>
<td></td>
<td>Convolution</td>
<td>Using a 1-dilated 4 x 4 filter</td>
<td>Conv2D</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>MatMul</td>
<td>transpose_a = False, transpose_b = False, adjoint_a = False, adjoint_b = False -</td>
<td>MatMulV2</td>
</tr>
<tr>
<td></td>
<td>Conv2D</td>
<td>-</td>
<td>Conv2D</td>
</tr>
<tr>
<td></td>
<td>DepthwiseConv2dNative</td>
<td>dilation = 1</td>
<td>DepthwiseConv2D</td>
</tr>
</tbody>
</table>

The fusion types implemented in quantization are as follows. (Currently, only BN fusion is supported.)

**Table 8-2** Fusion support

<table>
<thead>
<tr>
<th>Framework</th>
<th>Fusion Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td>Conv+BN+Scale fusion</td>
<td>Before quantization, the &quot;Convolution +BatchNorm+Scale&quot; composite in the model is fused into &quot;Conv+BN+Scale.&quot; The BatchNorm and Scale layers are removed.</td>
</tr>
</tbody>
</table>
### 8.1.3 Tool Workflow

**Table 8-3** Major actions in the tool workflow

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set up Ascend 910 environment</td>
<td>Set up the Ascend 910 development and operating environments and install components such as Driver, Firmware, and FwkACLlib by referring to <em>CANN Software Installation Guide</em> of Ascend 910 AI Processor.</td>
</tr>
</tbody>
</table>
### 8.2 AMCT Installation

#### 8.2.1 获取软件包

#### 8.2.2 Pre-installation Actions

**Preparing the AMCT User**

Perform the installation as the user who has installed the components in the Ascend 910 environment. This section uses a non-root user as an example.

For details about the permission settings of a non-root user, see section "Pre-installation Actions" in CANN Software Installation Guide of Ascend 910 AI Processor.

**Setting Up Environment**

**NOTE**

This tool supports Python 3.7.0 to Python 3.7.9. This document uses Python 3.7.5 as an example. The environment variables and installation commands vary according to the actual Python version.

AMCT needs to be installed in the Ascend 910 environment, which has the following hardware requirements.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install AMCT</td>
<td>Install AMCT by referring to <strong>8.2 AMCT Installation</strong>. Before the installation, you need to obtain the AMCT package, create an AMCT installation user, check the environments, install dependencies, and upload the AMCT package.</td>
</tr>
<tr>
<td>Run quantization</td>
<td>Run the <code>amct_acl</code> command to quantize the original model into an .air model by referring to <strong>8.3 Quantization</strong>.</td>
</tr>
</tbody>
</table>
| Run inference on the quantized model | 1. Set up the Ascend 310 inference environment (you are advised to set up the development and operating environments on the same server) by referring to CANN Software Installation Guide of Ascend 310 AI Processor and upload the quantized model to the environment.  
  2. Use the ATC tool to convert the .air quantized model into an .om offline model adapted to the Ascend AI Processor.  
  3. Run inference on the converted .om offline model. |
Table 8-4 Version mapping

<table>
<thead>
<tr>
<th>Item</th>
<th>Version</th>
<th>How to Obtain</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ascend 910 OS</td>
<td>Ubuntu 18.04 (x86_64)</td>
<td>See CANN Software Installation Guide of Ascend 910 AI Processor.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>EulerOS release 2.0 (SP9)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AArch64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td>3.7.5</td>
<td>The dependency is required for both training and inference. For the installation instructions, see &quot;Preparations for Installation&quot; in CANN Software Installation Guide.</td>
<td>-</td>
</tr>
<tr>
<td>Ascend 310 Inference OS</td>
<td>Ubuntu 18.04 (x86_64)</td>
<td>See CANN Software Installation Guide of Ascend 310 AI Processor.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>EulerOS release 2.0 (SP9)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AArch64</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Installing Dependencies

For details, see "Pre-installation Actions" in CANN Software Installation Guide of Ascend 910 AI Processor.

Uploading the AMCT Package

Upload the Ascend-amct-{software version}-{os}.{arch}.tar.gz package to any directory on the Linux server as the AMCT installation user. In this example, the package is uploaded to $HOME/amct/.

Decompress the AMCT package:

tar -zxvf Ascend-amct-{software version}-{os}.{arch}.tar.gz

Find the following extracted packages.

Table 8-5 Extracted parts of the AMCT package

<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
<tbody>
<tr>
<td>amct căfeffe/</td>
<td>Caffe AMCT directory</td>
<td>● OS support: Ubuntu 18.04 (x86_64)</td>
<td></td>
</tr>
<tr>
<td>Level-1 Directory</td>
<td>Level-2 Directory</td>
<td>Description</td>
<td>Use Case and Precaution</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------</td>
<td>------------------------</td>
</tr>
</tbody>
</table>
| **amct_caffe**    | `{version}-py3-none-linux_{arch}.whl` | Caffe AMCT package | • For details, see *AMCT Instructions (Caffe)*.  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| `sample.tar.gz`   |                    | Caffe quantization sample package | |
| `caffe_patch.tar.gz` |                  | Caffe patch package | |

**TensorFlow AMCT directory**

<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
</table>
| **amct_tensorflow** | `{version}-py3-none-linux_{arch}.whl` | TensorFlow AMCT package | • OS support: Ubuntu 18.04 (x86_64)  
• `amct_tensorflow` and `amct_tensorflow_ascend` cannot exist at the same time.  
• For details, see *AMCT Instructions (TensorFlow)*.  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| `sample.tar.gz` |                    | TensorFlow quantization sample package | |

<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
</table>
| **amct_tensorflow_ascend** | `{version}-py3-none-linux_{arch}.whl` | TF Adapter AMCT package | • OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
• `amct_tensorflow` and `amct_tensorflow_ascend` cannot exist at the same time.  
• For details, see *AMCT Instructions (TensorFlow, Ascend)*.  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
<p>| <code>sample.tar.gz</code> |                    | Package of quantization samples using TF Adapter | |</p>
<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
</table>
| **amct_pytorch/** | **PyTorch AMCT directory** | | • OS support: Ubuntu 18.04 (x86_64)  
• For details, see *AMCT Instructions (PyTorch)*.  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| | **amct_pytorch-\{version\}-py3-none-linux_{arch}.tar.gz** | PyTorch AMCT source package | |
| | **amct_pytorch_sample.tar.gz** | PyTorch quantization sample package | |
| **amct_onnx/** | **ONNX AMCT directory** | | • OS support: Ubuntu 18.04 (x86_64)  
• For details, see *AMCT Instructions (ONNX)*.  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| | **amct_onnx-\{version\}-py3-none-linux_{arch}.whl** | ONNX AMCT package | |
| | **amct_onnx_op.tar.gz** | ONNX Runtime AMCT custom OPP | |
| | **amct_onnx_sample.tar.gz** | ONNX quantization sample package | |
| **amct_ms/** | **MindSpore AMCT directory** | | • OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
• For details, see *AMCT Instructions (MindSpore)*.  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
| | **amct_mindspore-\{version\}-py3-none-linux_{arch}.whl** | MindSpore AMCT package | |
| | **amct_mindspore_sample.tar.gz** | MindSpore quantization sample package | |
| **amct_acl/** | **AscendCL API AMCT directory** | | • OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment  
• For details, see *AMCT Instructions (AscendCL)*.  
• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor. |
<p>| | <strong>Ascend-amct_acl-{software version}-{os}.{arch}.run</strong> | AscendCL API AMCT package | |</p>
<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>amct_acl_sample.tar.gz</td>
<td>Package of quantization samples using AscendCL APIs</td>
<td>installed with the Ascend AI Processor.</td>
</tr>
</tbody>
</table>

{version} indicates the AMCT version number. {os}.{arch} indicates the OS and architecture.

### 8.2.3 Installation

**Step 1** Switch to the directory where the AMCT runfile is stored and run the following command in the `amct/amct_acl` subdirectory to install AMCT:

```
./Ascend-amct_acl-{software version}-{os}.{arch}.run --full
```

For more available options, see 8.7.4 Supported Command-line Options for AMCT Installation or run the `./*.run --help` command.

The default installation path is $HOME/Ascend for a non-root user or /usr/local/Ascend for the root user. You can specify the path or overwrite a previous installation by using the `--install-path=path` option, in which `path` can be either absolute or relative.

- If the specified path does not exist, a directory is automatically created during the installation. If there are multiple levels of directories, the directory is automatically created only when the last level of directory does not exist.
- If the specified path exists, ensure that the installation user has the read, write, and execute permissions on the path.

**Step 2** Check the installation. If the following message is displayed, the installation is successful:

```
Amct_acl package install success!
```

- Default installation path:
  - For the root user: /usr/local/Ascend.
  - For a non-root user: $HOME/Ascend.
- Installation log path:
  - For the root user: /var/log/ascend_seclog/ascend_install.log.
  - For a non-root user: $HOME/var/log/ascend_seclog/ascend_install.log.
- File that records the installation path, installation mode, and running user information: $[install_path]/amct_acl/ascend_install.info

Replace $[install_path] with the actual installation path.

----End
8.2.4 Post-installation Actions

After the installation is complete, set the following environment variables for quantization:

- Add the path of the AMCT executable file to the PATH variable. Replace \$\{install\_path\} with the actual installation path of amct\_acl.
  
  ```bash
  export PATH=${install_path}/amct_acl/bin/:$PATH
  ```

- The following environment variables are optional:
  - For printing logs to the screen:
    ```bash
    export AMCT\_PRINT\_TO\_STDOUT=1
    ```
  - For exporting the quantization factors to a file:
    ```bash
    export DUMP\_AMCT\_RECORD=1
    ```

8.3 Quantization

**Step 1** Decompress the sample software package.

Go to the amct/amct\_acl directory and decompress the sample package:

```bash
  tar -zxvf amct\_acl_sample.tar.gz
  cd sample
```

Find the following extracted directories:

- **resnet50\_caffe**: quantization directory of the Caffe-based ResNet-50 network for image classification.

For details, see 8.7.1 Directory Structure of the Sample Package.

**Step 2** Run quantization.

- **Caffe-based ResNet-50 quantization**
  a. Go to the sample/resnet50\_caffe directory and run the following command to download the ResNet-50-deploy.prototxt model file:

  ```bash
  python3.7.5 download\_prototxt.py --close\_certificate\_verify
  ```

  The `--close\_certificate\_verify` option is optional. It is used to disable certificate verification to ensure that the model can be downloaded properly. If a message is displayed indicating that authentication fails during model download, include this option to your download command and try again.

  If messages similar to the following are displayed, the model file is successfully downloaded:

  ```bash
  ```

  You can inspect the downloaded model file in the sample/resnet50\_caffe/pre\_model directory as prompted.
  b. Prepare a calibration dataset.

  Run the following command to convert the .jpg images in the image directory into a binary dataset:

  ```bash
  python3.7.5 process\_data.py
  ```
After the command is executed, the `calibration.bin` dataset is generated in the `image` directory.

c. Run the following command in the `sample/resnet50_caffe` directory to perform quantization:

```
```

**NOTICE**

The `amct_acl` command can be organized in either of the following ways:

1. `amct_acl param1=value1 param2=value2 ...` (No space is allowed before `value`. Otherwise, `value` will be truncated, and the value of `param` is empty.)
2. `amct_acl param1 value1 param2 value2 ...`

For the available command-line options, see 8.4 Command-line Options. The sample provides the preceding commands and also an encapsulated script `run_calibration.sh` of the dataset preprocessing script described in Step 2.b. You can directly run this encapsulated script to perform quantization in the `sample/resnet50_caffe` directory with the following command:

```
bash run_calibration.sh
```

If a message similar to the following is displayed, the quantization is successful:

```
amct_acl generate deploy air success.
```

d. View the quantization result.

After successful quantization, find the following files generated to the `sample/resnet50_caffe` directory:

- `amct_log/amct_acl.log`: quantization log directory, recording the AMCT quantization logs.
- `kernel_meta`: directory of operator build outputs.
- `dump/record.txt`: (optional) quantization factor record file. If the environment variable for generating quantization factors is set during quantization (for details, see 8.2.4 Post-installation Actions), the directory is generated. For details about quantization factors, see 8.7.3 Quantization Factor Record File.

e. (Optional) Convert the quantized model into an offline model adapted to the Ascend AI Processor by referring to ATC Instructions.

Example:

```
atc --model=$HOME/module/ResNet-50-model.air --framework=1 --output=$HOME/module/out/ResNet-50_om --soc_version=${soc_version}
```

- **TensorFlow-based ResNet-50 quantization**

  a. Prepare a calibration dataset.
Run the following command to convert the .jpg images in the `image` directory into a binary dataset:

```bash
python3.7.5 process_data.py
```

After the command is executed, the `calibration.bin` dataset is generated in the `image` directory.

b. Run the following command in the `sample/resnet50_tf` directory to perform quantization:

```bash
amct_acl --framework=3 --model=./pre_model/resnet_v1_50.pb --calibration_data=./image/calibration.bin --calibration_shape="input:16,224,224,3" --output=./results/resnet_v1_50 --soc_version=Ascend910A --log=info
```

For the available command-line options, see 8.4 Command-line Options. The sample provides the preceding commands and also an encapsulated script `run_calibration.sh` of the dataset preprocessing script described in Step 2.a. You can directly run this encapsulated script to perform quantization in the `sample/resnet50_tf` directory with the following command:

```bash
bash run_calibration.sh
```

If a message similar to the following is displayed, the quantization is successful:

```
amct_acl generate deploy air success.
```

c. View the quantization result.

After successful quantization, find the following files generated to the `sample/resnet50_tf` directory:

- `amct_log/amct_acl.log`: quantization log directory, recording the AMCT quantization logs.
- `kernel_meta`: directory of operator build outputs.
- `results/resnet_v1_50.air`: quantized model file.
- `dump/record.txt`: (optional) quantization factor record file. If the environment variable for generating quantization factors is set during quantization (for details, see 8.2.4 Post-installation Actions), the directory is generated. For details about quantization factors, see 8.7.3 Quantization Factor Record File.

d. (Optional) Convert the quantized model into an offline model adapted to the Ascend AI Processor by referring to ATC Instructions.

Example:

```bash
atc --model=$HOME/module/resnet_v1_50.air --framework=1 --output=$HOME/module/out/resnet_v1_50.om --soc_version=${soc_version}
```

---End

8.4 Command-line Options
## 8.4.1 Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Option</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General option</strong></td>
<td><strong>8.4.2.1.1 -- help or --h</strong></td>
<td>No</td>
<td>Displays help information.</td>
</tr>
<tr>
<td><strong>Input options</strong></td>
<td><strong>8.4.2.2.1 -- model</strong></td>
<td>Yes</td>
<td>Sets the model file directory, including the file name.</td>
</tr>
<tr>
<td></td>
<td><strong>8.4.2.2.2 -- weight</strong></td>
<td>Required for Caffe model.</td>
<td>Sets the weight file directory, including the file name. This option is required for a Caffe model.</td>
</tr>
<tr>
<td></td>
<td><strong>8.4.2.2.3 -- framework</strong></td>
<td>Yes</td>
<td>Specifies the framework of the original model. Currently, this option can only be set to 0 (Caffe) or 3 (TensorFlow).</td>
</tr>
<tr>
<td></td>
<td><strong>8.4.2.2.4 -- input_form at</strong></td>
<td>No</td>
<td>Sets the model input format. Defaults to NHWC for TensorFlow or NCHW for Caffe.</td>
</tr>
<tr>
<td></td>
<td><strong>8.4.2.2.5 -- calibration _data</strong></td>
<td>Yes</td>
<td>Specifies the calibration data file of the model.</td>
</tr>
<tr>
<td></td>
<td><strong>8.4.2.2.6 -- calibration _shape</strong></td>
<td>Yes</td>
<td>Sets the calibration input node name and data shape of the model.</td>
</tr>
<tr>
<td></td>
<td><strong>8.4.2.2.7 -- calibration _config</strong></td>
<td>No</td>
<td>Sets the simplified quantization configuration file. If it is left empty, the default configuration is used.</td>
</tr>
<tr>
<td><strong>Output option</strong></td>
<td><strong>8.4.2.3.1 -- output</strong></td>
<td>Yes</td>
<td>Sets the name for the output .air file, which is automatically suffixed with the .air extension.</td>
</tr>
<tr>
<td><strong>Target SoC option</strong></td>
<td><strong>8.4.2.4.1 -- soc_version</strong></td>
<td>Yes</td>
<td>Currently, only Ascend910A and Ascend910B are supported.</td>
</tr>
<tr>
<td><strong>Feature options</strong></td>
<td><strong>8.4.3.1.1 -- out_nodes</strong></td>
<td>No</td>
<td>Sets the output nodes.</td>
</tr>
<tr>
<td></td>
<td><strong>8.4.3.1.2 -- input_fp16_nodes</strong></td>
<td>No</td>
<td>Sets the input nodes to specify as fp16 nodes.</td>
</tr>
<tr>
<td><strong>Debug option</strong></td>
<td><strong>8.4.3.2.1 -- log</strong></td>
<td>No</td>
<td>Sets the log level.</td>
</tr>
</tbody>
</table>
8.4.2 Basic Functionality

8.4.2.1 General Option

8.4.2.1.1 --help or --h

**Description**

Displays help information.

**See Also**

None

**Argument**

None

**Recommended Configurations and Benefits**

None

**Example**

```
amct_acl --help
```

**Dependencies and Restrictions**

None

8.4.2.2 Input Options

8.4.2.2.1 --model

**Description**

Sets the model file directory, including the file name.

**See Also**

For a Caffe model file, this option must work with 8.4.2.2.2 --weight.

**Argument**

*Argument*: model file directory, including the file name.

*Format*: The directory and file name can contain letters, digits, underscores (_), hyphens (-), and periods (.).
Recommended Configurations and Benefits
None

Example
--model=$HOME/test/resnet50.prototxt --weight=$HOME/test/resnet50.caffemodel

Dependencies and Restrictions
None

8.4.2.2.2 --weight

Description
Sets the weight file directory, including the file name.
This option is required for a Caffe model.

See Also
For a Caffe model file, this option must work with 8.4.2.2.1 --model.

Argument
Argument: weight file directory, including the file name.
Format: The directory and file name can contain letters, digits, underscores (_), hyphens (-), and periods (.)
Argument

**Argument:**

- 0: Caffe
- 3: TensorFlow

**Restrictions:** This option is required. Currently, only 0 or 3 is supported.

Recommended Configurations and Benefits

None

Example

- **Caffe:**
  
  ```
  ```

- **TensorFlow:**

  ```
  amct acl --framework=3 --model=./pre_model/resnet_v1_50.pb --calibration_data=./image/calibration.bin --calibration_shape="input:16,224,224,3" --output=./results/resnet_v1_50 --soc_version=Ascend910 --log=info
  ```

Dependencies and Restrictions

None

8.4.2.2.4 --input_format

**Description**

Sets the model input format.

**See Also**

None

**Argument**

**Argument:**

- If the original framework is Caffe, NCHW (default) or ND (any format with N \leq 4) is supported.
- If the original framework is TensorFlow, NCHW, NHWC (default), ND, NCDHW, or NDHWC is supported.

**Default:** NCHW for Caffe, or NHWC for TensorFlow.

Recommended Configurations and Benefits

None

**Example**

```
--input_format=NCHW
```
Dependencies and Restrictions

None

8.4.2.2.5 --calibration_data

Description

Sets the path of the calibration data file.

See Also

This option must work with 8.4.2.2.6 --calibration_shape.

Argument

Argument: name of the calibration data file.

Format: The directory and file name can contain letters, digits, underscores (\_), hyphens (-), and periods (.). Separate multiple files with commas (,).

Recommended Configurations and Benefits

None

Example

The following assumes that the model has multiple inputs, which are separated with commas (,). Set the option according to your model.

```
--calibration_data=./tf_fcn/keep_probability.bin,./tf_fcn/tmp_featuremap_input.bin --calibration_shape="input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2"
```

Dependencies and Restrictions

None

8.4.2.2.6 --calibration_shape

Description

Sets the shape of the calibration data.

See Also

This option must work with 8.4.2.2.5 --calibration_data.

Argument

Argument:

Calibration data shape, for example, "input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2". Enclose all nodes in double quotation marks (\"\") and separate the nodes with semicolons (;).
input_name must be the node name in the network model before model conversion.

Recommended Configurations and Benefits

None

Example

```
--calibration_shape="input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2"  --calibration_data=./tf_fcn/
keep_probability.bin,/tf_fcn/tmp_featuremap_input.bin
```

Dependencies and Restrictions

None

8.4.2.2.7 --calibration_config

Description

Sets the simplified quantization configuration file. For the definition of the file, see 8.7.2 Simplified Quantization Configuration File.

See Also

None

Argument

Argument: name of the simplified quantization configuration file.

Format: The directory and file name can contain letters, digits, underscores (_), hyphens (-), and periods (.)

Recommended Configurations and Benefits

None

Example

```
--calibration_config=quant.cfg
```

Dependencies and Restrictions

None

8.4.2.3 Output Option

8.4.2.3.1 --output

Description

Sets the directory and name for the output .air file.
See Also
None

Argument

**Argument**: directory and name of the output .air file. The file name is automatically suffixed with the .air extension.

**Format**: The directory and file name can contain letters, digits, underscores (_), hyphens (-), and periods (.). Separate multiple files with commas (,).

Recommended Configurations and Benefits
None

Example

```
--output=./results/resnet_v1_50
```

Dependencies and Restrictions
None

8.4.2.4 Target SoC Option

8.4.2.4.1 --soc_version

Description

Sets the SoC version used in quantization.

See Also
None

Argument

Currently, only **Ascend910A** and **Ascend910B** are supported.

Recommended Configurations and Benefits
None

Example

```
The following uses **Ascend910A** as an example.

--soc_version=Ascend910A
```

Dependencies and Restrictions
None
### 8.4.3 Advanced Functionality

#### 8.4.3.1 Feature Options

#### 8.4.3.1.1 --out_nodes

**Description**

Sets the output nodes.

**See Also**

None

**Argument**

**Argument:**

Name of the node (node_name) in the network model. Enclose all nodes in double quotation marks ("""") and separate the nodes with semicolons (;). node_name must be the node name of the original model. The number after the colon indicates the output index. For example, node_name1:0 indicates output 0 of the node named node_name1.

**Restrictions:**

1. The --out_nodes option is required only when there are output node changes in the original model. For example, the original output node may be removed or replaced due to quantization or operator fusion, as shown in Figure 8-3.

   **Figure 8-3 Output node changes**

   ![Output node changes diagram]

   • Output node change after quantization
   • Output node change after fusion

2. If the original model has multiple output nodes and the output nodes change, --out_nodes must be set to the names of all output nodes.

**Recommended Configurations and Benefits**

None

**Example**

```
--out_nodes="node_name1:0;node_name1:1;node_name2:0"
```
 Dependencies and Restrictions
   None

8.4.3.1.2 --input_fp16_nodes

Description
   Sets the input nodes to specify as fp16 nodes.

See Also
   None

Argument
   Argument: node names to specify as fp16 nodes.
   Restrictions: The specified nodes must be enclosed in double quotation marks ("") and separated by semicolons (;).

Recommended Configurations and Benefits
   None

Example
   --input_fp16_nodes="node_name1;node_name2"

Dependencies and Restrictions
   None

8.4.3.2 Debug Option

8.4.3.2.1 --log

Description
   Sets the log level during AMCT running.

See Also
   None

Argument
   Argument:
   ●   debug: prints debug, info, warning, error, and event logs.
   ●   info: prints info, warning, error, and event logs.
   ●   warning: prints warning, error, and event logs.
- **error**: prints error and event logs.

**Default**: info

**Restrictions**:

This option controls only the log level displayed by AMCT. To view the logs of other components, such as GE, find the corresponding log file or set the following environment variable to print the log to the screen:

```bash
export ASCEND_SLOG_PRINT_TO_STDOUT=1
```

You can also change log levels for other components using environment variables. For details, see Setting log levels for other components.

For details about logs of other components, see Log Reference.

**Recommended Configurations and Benefits**

None

**Example**

```bash
--log=info
```

**Dependencies and Restrictions**

- **Log flushing**:
  During the execution of the `amct_acl` command, logs are flushed by default (to the `amct_log` log directory after quantization). The default log level is **info**.

- **Log printing to the screen**:
  During the execution of the `amct_acl` command, logs are not printed by default. To print logs to the screen, set the following environment variable in the current window where the `amct_acl` command is executed:

```bash
export AMCT_PRINT_TO_STDOUT=1
```

- **Setting log levels for other components, such as GE**:
  - Setting the global log level
    The global log level set through the `ASCEND_GLOBAL_LOG_LEVEL` environment variable on the host side applies only to the global and module log level settings in the current shell. If the environment variable is not set or set to an invalid value, the ERROR level is used by default. You can run the `echo ASCEND_GLOBAL_LOG_LEVEL` command to view the current log level. If the log level is invalid or empty, the log level is the default level.
    
    See the following example.

```bash
export ASCEND_GLOBAL_LOG_LEVEL=1
```

**Value range of `ASCEND_GLOBAL_LOG_LEVEL`**:

- 0: DEBUG
- 1: INFO
- 2: WARNING
- 3: ERROR
- 4: NULL (no log output)
- Other values: invalid

Enabling event logging
The event logging mode set through the `ASCEND_GLOBAL_EVENT_ENABLE` environment variable applies only in the current shell. If the environment variable is not set or set to an invalid value, the event logging is enabled by default. You can run the `echo $ASCEND_GLOBAL_EVENT_ENABLE` command to view the current setting. If the no value or an invalid value is returned, the default setting is used.

See the following example.
```bash
export ASCEND_GLOBAL_EVENT_ENABLE=0
```

Value range of `ASCEND_GLOBAL_EVENT_ENABLE`:
- 0: disabled
- 1: enabled
- Other values: invalid

### 8.5 AMCT Upgrade

AMCT can be upgraded by overwriting an existing installation or by using the upgrade command. In both scenarios, the existing version is uninstalled before the new one is installed.

1. Upgrade AMCT.
   - By overwriting the existing version:
     ```bash
     ./Ascend-amct_acl-{software version}-{os}.{arch}.run --full
     ```
   - By using the upgrade command:
     ```bash
     ./Ascend-amct_acl-{software version}-{os}.{arch}.run --upgrade
     ```
     If an installation path is specified, the `--install-path=${install_path}` option must be included in the upgrade command.

     If the following information is displayed, the upgrade is successful:
     ```
     Amct_acl package upgrade success!
     ```

2. Check the version number after the upgrade.
   Go to the installation path of the runfile to be queried, for example, the defaulted path `/usr/local/Ascend/$runfile_name` for the root user, or the defaulted path `$HOME/Ascend/$runfile_name` for a non-root user. Then, run the following command to check the version:
   ```bash
   cat version.info
   ```

3. Inspect the upgrade log file.
   - For the root user: `/var/log/ascend_seclog/ascend_install.log`
   - For a non-root user: `$HOME/var/log/ascend_seclog/ascend_install.log`
8.6 AMCT Uninstallation

Uninstall AMCT in either of the following ways:

- By running the following command in any directory:
  ```bash
  bash $(install_path)/amct_acl/script/uninstall.sh
  ```
  Replace `$(install_path)` with the AMCT installation path.

- By using the runfile:
  Run the following command in the runfile directory:
  ```bash
  ./Ascend-amct_acl-{software version}-{os}.{arch}.run --uninstall
  ```
  If an installation path is specified, the `--install-path=$(install_path)` option must be included in the uninstallation command.

If the following information is displayed, the uninstallation is successful:

```
Amct_acl package uninstall success!
```

8.7 Appendixes

8.7.1 Directory Structure of the Sample Package

<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>classification.jpg</td>
<td>Calibration dataset used by the Caffe-based ResNet-50 network.</td>
</tr>
<tr>
<td></td>
<td>download_prototxt.py</td>
<td>Script for downloading the ResNet-50 model file.</td>
</tr>
<tr>
<td></td>
<td>image</td>
<td>Built-in image dataset of the Caffe-based ResNet-50 sample, containing 32 images.</td>
</tr>
<tr>
<td></td>
<td>process_data.py</td>
<td>Dataset preprocessing script.</td>
</tr>
<tr>
<td></td>
<td>run_calibration.sh</td>
<td>Encapsulated script for running quantization.</td>
</tr>
</tbody>
</table>
## 8.7.2 Simplified Quantization Configuration File

**Table 8-7 calibration_config.proto description**

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified post-training quantization configuration file of AMCT.</td>
</tr>
<tr>
<td>Op tional</td>
<td>bool</td>
<td>activation_offset</td>
<td></td>
<td>Whether to quantize activations with offset.</td>
</tr>
<tr>
<td>Re peated</td>
<td>string</td>
<td>skip_layers</td>
<td></td>
<td>Layers to skip during quantization.</td>
</tr>
<tr>
<td>Re peated</td>
<td>string</td>
<td>skip_layer_types</td>
<td></td>
<td>Types of layers to skip during quantization.</td>
</tr>
<tr>
<td>Op tional</td>
<td>int32</td>
<td>version</td>
<td></td>
<td>Version of simplified configuration file.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------</td>
<td>---------------------------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>CalibrationConfig</td>
<td>common_config</td>
<td>Common quantization configuration. If a layer is not overridden by <code>override_layer_types</code> or <code>override_layer_configs</code>, this configuration is used.</td>
</tr>
<tr>
<td>Repeated</td>
<td>override_layer_type</td>
<td>override_layer_configs</td>
<td></td>
<td>Type of the layer to override.</td>
</tr>
<tr>
<td>Repeated</td>
<td>override_layer</td>
<td>override_layer_configs</td>
<td></td>
<td>Layer to override.</td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>do_fusion</td>
<td></td>
<td>BN fusion switch. Defaults to <code>true</code>, indicating BN fusion enabled.</td>
</tr>
<tr>
<td>Repeated</td>
<td>string</td>
<td>skip_fusion_layers</td>
<td></td>
<td>Layers to skip during BN fusion.</td>
</tr>
<tr>
<td>OverrideLayerType</td>
<td>Required</td>
<td>string</td>
<td>layer_type</td>
<td>Types of layers supporting quantization.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>CalibrationConfig</td>
<td>calibration_config</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td>OverrideLayer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization configuration overriding by layer.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>layer_name</td>
<td>Layer to override.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>---------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>CalibrationConfig</td>
<td>Required</td>
<td>CalibrationConfig</td>
<td>calibration_config</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>ARQuantize</td>
<td>arq_quantize</td>
<td>Calibration-based quantization configuration.</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>FMRQuantize</td>
<td>ifmr_quantize</td>
<td>Weight quantization algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>arq_quantize</strong>: ARQ algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>ifmr_quantize</strong>: IFMR algorithm configuration.</td>
</tr>
<tr>
<td>ARQuantize</td>
<td>Optional</td>
<td>bool</td>
<td>channel_wise</td>
<td>ARQ quantization algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>channel_wise</strong>: Whether to use different quantization factors for each channel.</td>
</tr>
<tr>
<td>FMRQuantize</td>
<td>Optional</td>
<td>float</td>
<td>search_range_start</td>
<td>FMR quantization algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>search_range_start</strong>: Quantization factor search start.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>float</td>
<td>search_range_end</td>
<td><strong>search_range_end</strong>: Quantization factor search end.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>float</td>
<td>search_step</td>
<td><strong>search_step</strong>: Quantization factor search step.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
<td>--------</td>
<td>----------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>max_percentile</td>
<td>Upper search limit.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>min_percentile</td>
<td>Lower search limit.</td>
</tr>
</tbody>
</table>

The following is an example of the simplified configuration file `quant.cfg` constructed based on the file described above. **Optype** must be set to an Ascend IR–defined operator type. For details about the mapping, see Table 8-1.

```python
# global quantize parameter
activation_offset : true
version : 1
skip_layers : "conv_1"
skip_layer_types:"Optype"
do_fusion: true
skip_fusion_layers : "conv_1"
common_config : {
    arq_quantize : {
        channel_wise : true
    }
    ifmr_quantize : {
        search_range_start : 0.7
        search_range_end : 1.3
        search_step : 0.01
        max_percentile : 0.999999
        min_percentile : 0.999999
    }
}
override_layer_types : {
    layer_type : "Optype"
    calibration_config : {
        arq_quantize : {
            channel_wise : false
        }
        ifmr_quantize : {
            search_range_start : 0.8
            search_range_end : 1.2
            search_step : 0.02
            max_percentile : 0.999999
            min_percentile : 0.999999
        }
    }
}
override_layer_configs : {
    layer_name : "conv_2"
    calibration_config : {
        arq_quantize : {
            channel_wise : true
        }
    }
}
```
8.7.3 Quantization Factor Record File

Prototype

The quantization factor record file is a serialized data structure file based on Protobuf. You can generate a quantized model file by using the quantization configuration file, original network model file, and the quantization factor record file. The Protobuf prototype is defined as follows.

```protobuf
message SingleLayerRecord {
  optional float scale_d = 1;
  optional int32 offset_d = 2;
  repeated float scale_w = 3;
  repeated int32 offset_w = 4;
  repeated uint32 shift_bit = 5;
  optional uint32 channels = 6;
  optional uint32 height = 7;
  optional uint32 width = 8;
  optional bool skip_fusion = 9 [default = false];
}
message ScaleOffsetRecord {
  message MapFiledEntry {
    optional string key = 1;
    optional SingleLayerRecord value = 2;
  }
  repeated MapFiledEntry record = 1;
}
```

The parameters are described as follows.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScaleOffsetRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Map structure. To ensure compatibility, the discrete map structure is used.</td>
</tr>
</tbody>
</table>
| Repeated               |                    | MapFiledEntry | record      | Each records a quantization factor of a quantization layer and consists of two members:  
|                        |                    |                  |                          | • **key**: layer name.  
<p>|                        |                    |                  |                          | • <strong>value</strong>: quantization factor defined by SingleLayerRecord. |
| SingleLayerRecord      | -                 | -    | -           | Quantization factors for quantization.                                     |
| Repeated               | Optimal           | float | scale_d     | Scale factor for activation quantization. Only unified activation quantization is supported. |</p>
<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Optional</td>
<td>int32</td>
<td>offset _d</td>
<td>Offset factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>float</td>
<td>scale_ w</td>
<td>Scale factor for weight quantization. Scalar mode (quantizing the weight of the current layer in a unified manner) and vector mode (quantizing the weight of the current layer in channel-wise mode) are supported. Only Convolution and Deconvolution layers support channel-wise quantization.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>int32</td>
<td>offset _w</td>
<td>Offset factor for weight quantization. Similar to scale_w, it also supports the scalar and vector quantization modes and the dimension configuration must be the same as that of scale_w. Currently weight quantization with offset is not supported, and offset_w must be 0.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>uint32</td>
<td>shift_ bit</td>
<td>Shift factor. Reserved.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>uint32</td>
<td>channels</td>
<td>Network-wide infer_shape is not supported. Therefore, the input shape of the current layer needs to be configured. This field is used to configure the size of the input channel dimension.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>uint32</td>
<td>height</td>
<td>Network-wide infer_shape is not supported. Therefore, the input shape of the current layer needs to be configured. This field is used to configure the size of the input height dimension.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>uint32</td>
<td>width</td>
<td>Network-wide infer_shape is not supported. Therefore, the input shape of the current layer needs to be configured. This field is used to configure the size of the input width dimension.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------</td>
<td>------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>skip_fusion</td>
<td>Whether to skip Conv+BN+Scale fusion, Deconv+BN+Scale fusion, BN +Scale+Conv fusion, and FC+BN+Scale fusion at the current layer. Defaults to false, indicating that fusion of the preceding types is performed.</td>
</tr>
</tbody>
</table>

Beware that the Protobuf protocol does not report an error for repeated settings of optional fields. Instead, the most recent settings are used.

For general quantization layers, a range of parameters need to be configured, including `scale_d`, `offset_d`, `scale_w`, `offset_w`, `channels`, `height`, `width`, and `shift_bit`.

```plaintext
record {
  key: "conv1"
  value: {
    scale_d: 0.01424
    offset_d: -128
    scale_w: 0.43213
    scale_w: 0.78163
    scale_w: 1.03213
    offset_w: 0
    offset_w: 0
    offset_w: 0
    shift_bit: 1
    shift_bit: 1
    shift_bit: 1
    channels: 3
    height: 144
    width: 144
    skip_fusion: true
  }
}
record {
  key: "pool1"
  value: {
    scale_d: 0.532532
    offset_d: 13
    channels: 256
    height: 32
    width: 32
  }
}
record {
  key: "fc1"
  value: {
    scale_d: 0.37532
    offset_d: -67
    scale_w: 0.876221
    offset_w: 0
    shift_bit: 1
    channels: 1024
    height: 1
    width: 1
  }
}
```
Quantization Factors

The scale (a floating-point number) and offset quantization factors need to be provided for data and weight quantization. AMCT uses a unified quantization data structure. See the following expression.

\[ \text{data}_{\text{int8}} = \text{clip}_{\text{int8}}\left(\text{round}\left(\frac{\text{data}_{\text{float}}}{\text{scale}}\right)+\text{offset}\right) \]

The value ranges are as follows:

- \( \text{scale} \in [\text{FLT_EPSILON}, \frac{1}{\text{FLT_EPSILON}}] \), \( \text{FLT_EPSILON} \approx 1.1920929 \times 10^{-7} \)
- \( \text{offset} \in [-128, 127] \)

Quantization algorithms are classified into symmetric quantization algorithms and asymmetric quantization algorithms.

1. Symmetric quantization algorithm:
   The original high-precision data and quantized int8 data are converted into \( \text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int8}} \), where \( \text{scale} \) is a float32. To indicate positive and negative numbers, the signed int8 data type is used for \( \text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int8}} \). The following describes how to convert the original data into the int8 format. \text{round} is a rounding function. The value to be determined by the quantization algorithm is the constant \( \text{scale} \).

\[ \text{data}_{\text{int8}} = \text{round}\left(\frac{\text{data}_{\text{float}}}{\text{scale}}\right) \]

Quantization of the weights and activations may be summarized as a process of searching for a scale. Because \( \text{data}_{\text{int8}} \) is a signed number, to ensure symmetry of the ranges represented by positive and negative values, an absolute value operation is first performed on all data, so that the range of the to-be-quantized data is changed to \( \text{data}_{\text{int8}} \), and then \( \text{scale} \) is determined. The range of positive int8 values is [0, 127]. Therefore, \( \text{scale} \) can be computed as follows:

\[ \text{scale} = \frac{\text{data}_{\text{max}}}{127} \]

Therefore, the range of the int8 values is \([-128 \times \text{scale}, 127 \times \text{scale}]\).

Data beyond the range \([-128 \times \text{scale}, 127 \times \text{scale}]\) is saturated to a boundary value, and then the quantization operation shown in the formula is performed.

2. Asymmetric quantization algorithm:
   The difference from symmetric quantization algorithms lies in the data conversion mode. The \( \text{scale} \) and \( \text{offset} \) constants also need to be determined.

\[ \text{data}_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset}) \]

The uint8 data is converted through calculation based on the original high-accuracy data, which is shown in the following formula:

\[ \text{data}_{\text{uint8}} = \text{round}\left(\frac{\text{data}_{\text{float}}}{\text{scale}} - \text{offset}\right) \]
scale is an fp32, data\textsubscript{uint8} is an unsigned int8, and offset is an int8. The data range is \([\text{scale} \times \text{offset}, \text{scale} \times (255 + \text{offset})]\). If a value range of the to-be-quantized data is \([\text{data}_{\text{min}}, \text{data}_{\text{max}}]\), scale and offset are computed as follows:

\[
\text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}, \quad \text{offset} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}
\]

Unified quantization data format: By performing simple data conversion of the asymmetric quantization algorithm formula, the quantized data and the symmetric quantization algorithm are of the same type, that is, int. The specific conversion process is as follows:

int8 quantization is used as an example for description. The input original floating-point data is \(\text{data}_{\text{float}}\), the original quantized fixed-point number is \(\text{data}_{\text{float}}\), the quantization scale is \(\text{scale}\), and the quantization offset is \(\text{data}_{\text{float}}\) (the algorithm requires zero crossing to avoid too much accuracy drop). The calculation principle of quantization is as follows:

\[
\text{data}_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset}) = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset'} + 128) = \text{scale} \times (\text{data}_{\text{uint8}} - \text{offset'})
\]

Where,

\[
\text{data}_{\text{int8}} = \text{data}_{\text{uint8}} - 128 \in [-128, 127], \quad \text{offset'} = -(\text{offset} + 128) \in [-128, 127]
\]

Through the foregoing conversion, the data may also be converted into the int8 format. After scale and the converted offset' are determined, the int8 data converted from the original floating-point data is as follows:

\[
\text{data}_{\text{int8}} = \text{clip}\left(\text{round}\left(\frac{\text{data}_{\text{float}}}{\text{scale}}\right) + \text{offset'}\right)
\]

### 8.7.4 Supported Command-line Options for AMCT Installation

One-click installation is supported in the command line. You can select options as required to complete the installation. All options below are optional.

Installation command syntax: 

\[./\ast .\text{run} [\text{options}]\]

Table 8-8 describes the supported options.

**Table 8-8 Supported command-line options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--run</td>
<td>Run mode: installs only the files required in the AMCT running scenario.</td>
</tr>
<tr>
<td>--full</td>
<td>Full installation mode: installs all files.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--install-username=&lt;username&gt;</td>
<td>Initial installation: You can specify the running user.</td>
</tr>
<tr>
<td></td>
<td>Overwrite installation: The user name used in the last installation applies.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong></td>
</tr>
<tr>
<td></td>
<td>- This option must work with --install-usergroup=&lt;usergroup&gt;, and the value of username must be the one created in Preparing the AMCT User.</td>
</tr>
<tr>
<td></td>
<td>- If the root user is specified as the running user, the --install-for-all option must be included in the installation command as follows. In this scenario, security risks may exist.</td>
</tr>
<tr>
<td></td>
<td>--install-username=root --install-usergroup=root --install-for-all</td>
</tr>
<tr>
<td>--install-usergroup=&lt;usergroup&gt;</td>
<td>Initial installation: You can specify the running user group.</td>
</tr>
<tr>
<td></td>
<td>Overwrite installation: The user group used in the last installation applies.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong></td>
</tr>
<tr>
<td></td>
<td>This option must work with --install-username=&lt;username&gt;, and the value of usergroup must be the one created in Preparing the AMCT User.</td>
</tr>
<tr>
<td>--install-path=&lt;path&gt;</td>
<td>Sets the AMCT installation path. If the path is not specified, a default path applies:</td>
</tr>
<tr>
<td></td>
<td>- For the root user: /usr/local/Ascend.</td>
</tr>
<tr>
<td></td>
<td>- For a non-root user: $HOME/Ascend.</td>
</tr>
<tr>
<td></td>
<td>The running user must have the read and write permissions on the specified path. The path can contain letters, digits, underscores (_), hyphens (-), periods (.), and slashes (/). Note that periods (.) are not allowed in a relative path and slashes (/) are not allowed in a file or directory name.</td>
</tr>
<tr>
<td>--install-for-all</td>
<td>Allows all users to have the same installation group permission.</td>
</tr>
<tr>
<td></td>
<td>If this option is included in the installation or upgrade command, all users have the same permission on the directories and files created by the runfile installer as the installation group.</td>
</tr>
<tr>
<td></td>
<td>This option must work with any one among --run, --full, and --upgrade, for example, */run --full --install-for-all.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong></td>
</tr>
<tr>
<td></td>
<td>Make sure the security risks are considered before you include this option.</td>
</tr>
<tr>
<td>--uninstall</td>
<td>Uninstalls AMCT.</td>
</tr>
<tr>
<td>--noexec</td>
<td>Skips the execution of the installation script. Use this option with --extract=path. Format: --noexec --extract=path</td>
</tr>
<tr>
<td>--extract=path</td>
<td>Extracts the runfile to a specified path.</td>
</tr>
<tr>
<td>--upgrade</td>
<td>Upgrades AMCT.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--help or -h</td>
<td>Displays help information.</td>
</tr>
<tr>
<td>--check</td>
<td>Checks the runfile integrity and the mapping between runfiles.</td>
</tr>
<tr>
<td>--version</td>
<td>Queries runfile version.</td>
</tr>
<tr>
<td>--tar arg1 [arg2 ...]</td>
<td>Runs the <code>tar</code> command on the runfile. Use the arguments following <code>tar</code> as the command arguments. For example, the <code>--tar xvf</code> command decompresses a package to the current path.</td>
</tr>
<tr>
<td>--list</td>
<td>Lists the files in the runfile.</td>
</tr>
<tr>
<td>--info</td>
<td>Displays detailed information of a package.</td>
</tr>
<tr>
<td>--quiet</td>
<td>Performs silent installation with no further user input and minimal command-line output. In silent installation mode, if the permission on the installation directory is greater than 755, security risks are reported. In this case, you can choose to terminate the installation or re-execute the runfile in another directory with permission 755. This option must work with any one among <code>--run</code>, <code>--full</code>, <code>--upgrade</code>, and <code>--uninstall</code>, for example, <code>./*.run --full --quiet</code>.</td>
</tr>
</tbody>
</table>
9 AMCT Instructions (TensorFlow, Ascend 910)

9.1 Introduction

9.1.1 Overview

This document describes how to quantize a TensorFlow model using Ascend Model Compression Toolkit (AMCT). In the quantization process, the precision of model weights and activations is reduced to make the model more compact, improving the compute efficiency while lowering the transfer latency.

AMCT is a TensorFlow-based Python toolkit that requires a TF Adapter installation and implements layer fusion and 8-bit quantization of data and weights in neural networks. This toolkit decouples quantization from model conversion. It implements independent quantization of quantization-capable layers in a model and saves the quantized model to a .pb file. The obtained accuracy simulation model can run on CPU or NPU to complete accuracy simulation. The obtained deployable model can run on the Ascend AI Processor with improved inference performance. Currently, AMCT supports only the fp32 data type. The quantized model can serve for both accuracy simulation and inference deployment. This tool has the following advantages:

- Lightweight: You only need to install the tool.
- Easy-to-use APIs: You can complete quantization using APIs based on a TensorFlow inference script and a TF Adapter installation.
- Hardware compatibility: The quantized model can be converted to an offline model by using the ATC tool to implement 8-bit inference on the Ascend AI Processor.
- Configurable quantization: You can modify the quantization configuration file and adjust the quantization strategy to obtain the optimal quantization result.

The quantized TensorFlow network based on a TF Adapter installation can be used to implement inference on NPU or CPU. Figure 9-1 shows the deployment architecture. This document focuses only on the NPU inference scenario.

Figure 9-1 Architecture

9.1.2 Features

9.1.2.1 Terminology

There are two forms of quantization: post-training quantization and quantization aware training. Currently, only post-training quantization is supported.

Post-training quantization refers to quantizing the weights of an already-trained model from float32 to int8 and calibrating and quantizing the activations by using a small calibration dataset. For details about the quantization workflow, see 9.3 Post-training Quantization. (It is not supported to run a post-training quantization on more than one NPU.) Post-training quantization is classified into weight quantization and activation quantization according to different quantization objects. As used in this document, the following terms have the meanings specified below.

- Calibration dataset
  During the calibration process, the algorithm uses each piece of data in the calibration dataset as input, accumulates the input data of each layer (or operator) that needs to be quantized, and uses the accumulated data as input data of the quantization algorithm to determine the quantization factors. Because the quantization factors and the accuracy of the quantized model are closely related to the selection of the calibration dataset, you are advised to calibrate the model using a subset of images from the validation dataset.
• **Activation quantization**

Activation quantization is to collect statistics on the input data of each layer (or operator) to be quantized, to find the optimal pair of scale and offset per layer (or operator). For details, see [9.7.3 Quantization Factor Record File](#).

Activations are the intermediate results of model inference computation. The value ranges are input-specific. Therefore, a group of reference inputs (a calibration dataset) needs to be used as incentives to record the input data of each layer (or operator) to be quantized for searching for quantization factors (scale and offset).

• **Weight quantization**

After model training, the weights and the value ranges are determined. Therefore, quantization may be directly performed based on the value range of each weight.

The layers that support post-training quantization are listed as follows.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Restrictions</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatMul</td>
<td>transpose_a=False, transpose_b=False, adjoint_a=False, adjoint_b=False</td>
<td>The input source of weight does not contain nodes that can be dynamically changed, such as placeholder, and the node type of weight can only be const.</td>
</tr>
<tr>
<td>Conv2D</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DepthwiseConv2dNative</td>
<td>dilation=1</td>
<td></td>
</tr>
<tr>
<td>Conv2DBackpropInput</td>
<td>dilation=1</td>
<td></td>
</tr>
<tr>
<td>AvgPool</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

### 9.1.2.2 Implementation Principles

![Figure 9-2](image.png)

Figure 9-2 shows the AMCT principles. The operations in blue are implemented by users, and the operations in gray are implemented by using the AMCT API calls. You can import the library to the original TensorFlow network inference code and use the API calls at specific locations to implement the quantization function. The tool can be used in the following scenarios. This document mainly describes the NPU inference scenario. For the workflow and sample usage of the CPU inference scenario, see *AMCT Instructions (TensorFlow)* of Ascend 310 AI Processor.

- Run inference with the quantized model on NPU:
  - a. Construct an original TensorFlow model and then generate a quantization configuration file by using the [9.6.2.1 create_quant_config_ascend](#) call.
  - b. Optimize the TensorFlow model by using the [9.6.2.2 quantize_model_ascend](#) call based on the quantization configuration file. The optimized model contains the quantization algorithm. Run inference with the optimized model in the NPU environment based on the image dataset and calibration dataset preset in AMCT.
The image dataset is used to analyze the accuracy of the quantized data in the NPU environment. The calibration dataset is used to generate quantization factors to ensure accuracy.

c. Save the quantized model which can serve for both accuracy simulation on NPU and deployment on the Ascend AI Processor by using the 9.6.2.3 save_model_ascend call.

- Run inference with the quantized model on CPU:
  - Scenario 1
    i. Construct an original TensorFlow model and then generate a quantization configuration file by using the 9.6.2.1 create_quant_config_ascend call.
    ii. Optimize the TensorFlow model by using the 9.6.2.2 quantize_model_ascend call based on the quantization configuration file. The optimized model contains the quantization algorithm. Run inference with the optimized model in the TensorFlow environment based on the image dataset and calibration dataset preset in AMCT.

    The image dataset is used to analyze the accuracy of the quantized data in the TensorFlow environment. The calibration dataset is used to generate quantization factors to ensure accuracy.

    iii. By using the 9.6.2.3 save_model_ascend call, save the quantized model which can serve for both accuracy simulation in the TensorFlow environment and deployment on Ascend AI Processor.

    For details about APIs, see 9.6 API Description.

  - Scenario 2

    Instead of using the APIs in scenario 1, if you have generated a quantized model based on your own quantization factors and original TensorFlow model, complete the quantization by using the 9.6.2.7 convert_model call.

    Figure 9-2 Tool principles

9.1.3 Fusion Support

Currently, this tool mainly implements the following forms of layer fusion:
• Conv+BN fusion: Before AMCT quantization, the "Conv2D+BatchNorm" composite in the model is fused into "Conv+BN." The BatchNorm layer is removed.

• Depthwise_Conv+BN fusion: Before AMCT quantization, the "DepthwiseConv2dNative+BatchNorm" composite in the model is fused into "Depthwise_Conv+BN." The BatchNorm layer is removed.

• OP+(BiasAdd)+Mul fusion: Before AMCT quantization, the "Conv2D/MatMul/DepthwiseConv2dNative/Conv2DBackpropInput+Mul" and "Conv2D/MatMul/DepthwiseConv2d/Conv2DBackpropInput+BiasAdd+Mul" composites in the model are fused into "OP+(BiasAdd)+Mul." After the fusion, the Mul layer is removed.

  In this scenario, the other input of Mul must be of the Const type with an empty shape.

• Group_conv+BN fusion: If the "Split+Multi-Conv2D+ConcatV2 (or Concat, with concatenation performed along the C dimension)" composite is used in the model to indicate Group_conv, the "Group_conv+BatchNorm" composite is fused before AMCT quantization. The BatchNorm layer is removed.

  BN fusion applies to the following operators: FusedBatchNorm, FusedBatchNormV2, and FusedBatchNormV3.

• Small BN operators are fused into FusedBatchNormV3 on the condition that the small BN operators take 4D inputs. This fusion is supported only by post-training quantization.

  AMCT analyzes the composite of the small BN operators generated by \texttt{tf.keras.layers.BatchNormalization}, and replaces the small BN operators with larger BN composite on the following conditions:

  – On \texttt{tf.keras.layers.BatchNormalization} with fused=False and training=False, the network structures before and after fusion are as follows.
- On `tf.keras.layers.BatchNormalization` with `fused=False`, `center=False`, and `training=False`, the network structures before and after fusion are as follows.

- On `tf.keras.layers.BatchNormalization` with `fused=False`, `scale=False`, and `training=False`, the network structures before and after fusion are as follows.

- On `tf.keras.layers.BatchNormalization` with `fused=False`, `scale=False`, `center=False`, and `training=False`, the network structures before and after fusion are as follows.
9.1.4 Tool Workflow

Figure 9-3 shows the tool workflow.

**Figure 9-3 Tool workflow**

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set up Ascend 910 environment</td>
<td>Set up the Ascend 910 development and operating environments and install components such as Driver, Firmware, and FwkACLlib by referring to <em>CANN Software Installation Guide</em> of Ascend 910 AI Processor.</td>
</tr>
<tr>
<td>Action</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>Install the CPU version of TensorFlow</td>
<td>Ascend 910 supports only NPU quantization and does not support GPU quantization. Therefore, only the CPU version of TensorFlow needs to be installed. For details, see <a href="#">Installing Dependencies</a>.</td>
</tr>
<tr>
<td>Install AMCT</td>
<td>Install the TensorFlow version AMCT by referring to <a href="#">9.2 AMCT Installation</a>. Before the installation, you need to obtain the AMCT package, create an AMCT user, check the environment, install dependencies, and upload the AMCT package.</td>
</tr>
<tr>
<td>Run quantization</td>
<td>Run the provided quantization script to quantize an original network with the prepared dataset. For details, see <a href="#">9.3 Post-training Quantization</a>.</td>
</tr>
</tbody>
</table>
| (Optional) Run inference on the quantized model | 1. Set up an inference environment (the scenario where the development environment and operating environment are co-deployed is recommended) by referring to *CANN Software Installation Guide of Ascend 310 AI Processor* (inference purposes) and upload the quantized model to the environment.  
2. Use the ATC tool to convert the quantized model by referring to *[ATC Instructions](#)* of Ascend 310 AI Processor (inference purposes), and then run inference on the converted model. |

### 9.2 AMCT Installation

#### 9.2.1 获取软件包

#### 9.2.2 Pre-installation Actions

**Preparing the AMCT User**

Perform the installation as the user who has installed the components in the Ascend 910 environment. This section uses a non-root user as an example.

For details about the permission settings of a non-root user, see section "Pre-installation Actions" in *CANN Software Installation Guide of Ascend 910 AI Processor*.

**Setting Up Environment**

**NOTE**

This tool supports Python 3.7.0 to Python 3.7.9. This document uses Python 3.7.5 as an example. The environment variables and installation commands vary according to the actual Python version.
AMCT needs to be installed in the Ascend 910 environment, which has the following hardware requirements.

### Table 9-3 Version mapping

<table>
<thead>
<tr>
<th>Item</th>
<th>Version</th>
<th>How to Obtain</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ascend 910 OS</td>
<td>Ubuntu 18.04 (x86_64), EulerOS release 2.0 (SP9) AArch64</td>
<td>See <em>CANN Software Installation Guide</em> of Ascend 910 AI Processor.</td>
<td>-</td>
</tr>
<tr>
<td>Python</td>
<td>3.7.5</td>
<td>Install the dependencies in both the Ascend 910 and Ascend 310 environments by referring to the &quot;Pre-installation Actions&quot; section of the corresponding OS in <em>CANN Software Installation Guide</em>.</td>
<td>-</td>
</tr>
<tr>
<td>CPU version of TensorFlow</td>
<td>1.15</td>
<td>See <strong>Installing Dependencies</strong>.</td>
<td>-</td>
</tr>
<tr>
<td>Pillow</td>
<td>6.0.0+</td>
<td>See <strong>Installing Dependencies</strong>.</td>
<td>-</td>
</tr>
<tr>
<td>Ascend 310 OS</td>
<td>Ubuntu 18.04 (x86_64), EulerOS release 2.0 (SP9) AArch64</td>
<td>See <em>CANN Software Installation Guide</em> of Ascend 310 AI Processor. This environment is required for converting the quantized model into an offline model adapted to the Ascend AI Processor and running inference with the offline model.</td>
<td>-</td>
</tr>
</tbody>
</table>

### Installing Dependencies

For details, see "Pre-installation Actions" in *CANN Software Installation Guide* of Ascend 910 AI Processor. After the AMCT installation is complete, install Pillow, which is required for image processing.
### Table 9-4 Dependency list

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Version</th>
<th>Installation Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU version of TensorFlow</td>
<td>1.15</td>
<td>python3.7.5 -m pip install tensorflow-cpu==1.15 --user</td>
</tr>
<tr>
<td>Pillow</td>
<td>6.0.0+</td>
<td>pip3.7.5 install pillow==6.0.0 --user</td>
</tr>
</tbody>
</table>

### Uploading the AMCT Package

Upload the **Ascend-amct-{software version}-{os}.{arch}.tar.gz** package to any directory on the Linux server as the AMCT installation user. In this example, the package is uploaded to **$HOME/amct/**.

Decompress the AMCT package:

```
tar -zxvf Ascend-amct-{software version}-{os}.{arch}.tar.gz
```

Find the following extracted packages.

### Table 9-5 Extracted parts of the AMCT package

<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
<th>Use Case and Precaution</th>
</tr>
</thead>
<tbody>
<tr>
<td>amct_caffe/</td>
<td>Caffe AMCT directory</td>
<td></td>
<td>• OS support: Ubuntu 18.04 (x86_64)</td>
</tr>
<tr>
<td></td>
<td>amct_caffe-{version}-py3-none-linux_{arch}.whl</td>
<td>Caffe AMCT package</td>
<td>• For details, see AMCT Instructions (Caffe).</td>
</tr>
<tr>
<td></td>
<td>amct_caffe_sample.tar.gz</td>
<td>Caffe quantization sample package</td>
<td>• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor.</td>
</tr>
<tr>
<td></td>
<td>caffe_patch.tar.gz</td>
<td>Caffe patch package</td>
<td></td>
</tr>
<tr>
<td>amct_tensorflow/</td>
<td>TensorFlow AMCT directory</td>
<td></td>
<td>• OS support: Ubuntu 18.04 (x86_64)</td>
</tr>
<tr>
<td></td>
<td>amct_tensorflow-{version}-py3-none-linux_{arch}.whl</td>
<td>TensorFlow AMCT package</td>
<td>• <strong>amct_tensorflow</strong> and <strong>amct_tensorflow_ascend</strong> cannot exist at the same time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• For details, see AMCT Instructions (TensorFlow).</td>
</tr>
<tr>
<td>Level-1 Directory</td>
<td>Level-2 Directory</td>
<td>Description</td>
<td>Use Case and Precaution</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
<td>--------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>amct_tensorflow_sample.tar.gz</td>
<td>TensorFlow quantization package</td>
<td>• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor.</td>
</tr>
<tr>
<td></td>
<td>amct_tensorflow_ascend-{version}-py3-none-linux_{arch}.whl</td>
<td>TF Adapter AMCT package</td>
<td>• OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment</td>
</tr>
<tr>
<td></td>
<td>amct_tensorflow_ascend_sample.tar.gz</td>
<td>Package of quantization samples using TF Adapter</td>
<td>• <em>amct_tensorflow</em> and <em>amct_tensorflow_ascend</em> cannot exist at the same time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• For details, see <em>AMCT Instructions (TensorFlow, Ascend)</em>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor.</td>
</tr>
<tr>
<td></td>
<td>amct_pytorch/</td>
<td>PyTorch AMCT directory</td>
<td>• OS support: Ubuntu 18.04 (x86_64)</td>
</tr>
<tr>
<td></td>
<td>amct_pytorch-{version}-py3-none-linux_{arch}.tar.gz</td>
<td>PyTorch AMCT source package</td>
<td>• For details, see <em>AMCT Instructions (PyTorch)</em>.</td>
</tr>
<tr>
<td></td>
<td>amct_pytorch_sample.tar.gz</td>
<td>PyTorch quantization sample package</td>
<td>• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor.</td>
</tr>
<tr>
<td></td>
<td>amct_onnx/</td>
<td>ONNX AMCT directory</td>
<td>• OS support: Ubuntu 18.04 (x86_64)</td>
</tr>
<tr>
<td></td>
<td>amct_onnx-{version}-py3-none-linux_{arch}.whl</td>
<td>ONNX AMCT package</td>
<td>• For details, see <em>AMCT Instructions (ONNX)</em>.</td>
</tr>
<tr>
<td></td>
<td>amct_onnx_op.tar.gz</td>
<td>ONNX Runtime AMCT custom OPP</td>
<td>• Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor.</td>
</tr>
<tr>
<td>Level-1 Directory</td>
<td>Level-2 Directory</td>
<td>Description</td>
<td>Use Case and Precaution</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------</td>
<td>------------------------</td>
</tr>
<tr>
<td></td>
<td>amct_onnx_sample.tar.gz</td>
<td>ONNX quantization sample package</td>
<td>● OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment&lt;br&gt; ● For details, see AMCT Instructions (MindSpore).&lt;br&gt; ● Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor.</td>
</tr>
<tr>
<td>amct_ms/</td>
<td>MindSpore AMCT directory</td>
<td></td>
<td>● OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment&lt;br&gt; ● For details, see AMCT Instructions (MindSpore).&lt;br&gt; ● Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor.</td>
</tr>
<tr>
<td></td>
<td>amct_mindspore-{version}-py3-none-linux_{arch}.whl</td>
<td>MindSpore AMCT package</td>
<td></td>
</tr>
<tr>
<td></td>
<td>amct_mindspore_sample.tar.gz</td>
<td>MindSpore quantization sample package</td>
<td></td>
</tr>
<tr>
<td>amct_acl/</td>
<td>AscendCL API AMCT directory</td>
<td></td>
<td>● OS support: Ubuntu 18.04 (x86_64) in the Ascend 910 environment&lt;br&gt; ● For details, see AMCT Instructions (AscendCL).&lt;br&gt; ● Inference on a quantized model needs to be performed in the Ascend 310 inference environment installed with the Ascend AI Processor.</td>
</tr>
<tr>
<td></td>
<td>Ascend-amct_acl-{software version}-{os}.{arch}.run</td>
<td>AscendCL API AMCT package</td>
<td></td>
</tr>
<tr>
<td></td>
<td>amct_acl_sample.tar.gz</td>
<td>Package of quantization samples using AscendCL APIs</td>
<td></td>
</tr>
</tbody>
</table>

{version} indicates the AMCT version number. {os}.{arch} indicates the OS and architecture.

### 9.2.3 Installation

**Step 1**  In the directory where the AMCT package is located, run the following command:
```bash
pip3.7.5 install amct_tensorflow_ascend-{version}-py3-none-linux_{arch}.whl --user
```

Replace {version} with the actual AMCT version number, and {arch} with the actual architecture of the installation server. If AMCT installation is performed by the root user and the --target option is included, ensure that the path specified by --target is the path of the current user.

**Step 2**  Check the installation. If a message similar to the following is displayed, the installation is successful:
```
Successfully installed amct-tensorflow-ascend-{version}
```

Find the installed AMCT in the python3.7.5 directory (for example, $HOME/.local/lib/python3.7.5/site-packages).
9.3 Post-training Quantization

9.3.1 Quantizing an Image Classification Network

9.3.1.1 Quantization Preparations

Prerequisites

- **Model**
  Upload the TensorFlow model to be quantized to any directory on the Linux server as the AMCT installation user. This section uses the `mobilenetv2/pre_model/mobilenet_v2.pb` network model available in the sample package as an example.

  **NOTE**

  If you choose to use your own model, you are advised to perform inference in the TensorFlow environment in advance to test if it can run properly on the NPU in the TensorFlow environment with expected accuracy.

- **Image dataset**
  After the model is quantized using AMCT, perform inference with the model to test the model accuracy. Use the dataset that matches the model.
  Upload the dataset matching the model to any directory on the Linux server as the AMCT installation user.

- **Calibration dataset**
  The calibration dataset is used to generate the quantization factors to guarantee the accuracy.
  The process of calculating the quantization factors is referred to as calibration. Perform inference using the quantized network with a batch of a subset of images from the validation dataset to complete calibration.
  Upload the calibration dataset file to any directory on the Linux server as the AMCT installation user.

This section uses the MobileNetV2 network available in the sample package as an example.

9.3.1.2 Quantization Example

The following uses the image classification network’s quantization script `mobilenetv2_calibration.py`, the `pre_model/mobilenet_v2.pb` model file, the `image/classification.jpg` dataset, and the `image` calibration dataset to illustrate how to execute the quantization script.
1. Obtain the quantization script.

In the directory of `amct_tensorflow_ascend_sample.tar.gz`, extract the quantization script from the package:

```
tar -zxvf amct_tensorflow_ascend_sample.tar.gz
```

```
cd ascend_sample
```

Find the following extracted directories:

- `mobilenetv2/`: quantization directory of the image classification network MobileNetV2. This section describes how to execute the quantization script for image classification network.
- `yolov3/`: YOLOv3 quantization directory. For details, see 9.3.2 Quantizing an Object Detection Network.

For details about the directory structure, see 9.7.1 Sample Package Directory Structure.

2. Run the quantization script.

Run the following command in the `ascend_sample/mobilenetv2` directory to quantize the MobileNetV2 network:

```
python3.7.5 mobilenetv2_calibration.py [-h] [--cfg_define CFG_DEFINE]
```

Table 9-6 describes the command-line options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--h</td>
<td>(Optional) Displays help information.</td>
</tr>
<tr>
<td>--cfg_define CFG_DEFINE</td>
<td>(Optional) Sets the directory of the simplified quantization configuration file.</td>
</tr>
</tbody>
</table>

An example is as follows:

```
python3.7.5 mobilenetv2_calibration.py
```

If messages similar to the following are displayed, the quantization is successful:

```
INFO - [AMCT]:[save_model]: The model is saved in $HOME/amct/amct_tf/ascend_sample/mobilenetv2/result/MobileNetV2_ascend_quantized.pb     //Name and directory of the quantized model. It can both serves for accuracy simulation in the TensorFlow environment and runs on the Ascend AI Processor.
```

```
... Origin Model Prediction:  
  category index: 699  
  category prob: 0.559 //Inference result of the original model. It is an example only.
```

```
Quantized Model Prediction:  
  category index: 699  
  category prob: 0.611 //Inference result of the quantized model. It is an example only.
```

3. View the quantization result.

After the quantization is complete, find the following files generated in the directory of the quantization script:

- `calibration_tmp`:
  - `config_ascend.json`: configuration file that describes how to quantize each layer in the model. If a quantization configuration file already exists in the directory of the quantization script, the existing quantization configuration file is overwritten by a new one with the
same name in a call to 9.6.2.1 create_quant_config_ascend. If not that case, a new quantization configuration file is created.

If the accuracy of model inference drops significantly after quantization, you can modify the config_ascend.json file by referring to 9.3.3 Quantization Configuration.

- record_ascend.txt: quantization factor record file. For details about the prototype definition of the file, see 9.7.3 Quantization Factor Record File.


- tmp:
  - fusion_result.json: file generated when the TF Adapter is called for online inference on the NPU, which stores the graph fusion and UB fusion patterns.
  - check_result.tf.json: file generated when the TF Adapter is called for online inference on the NPU, which stores the operators that cannot run on the NPU in the inference graph.
  - kernel_meta: directory of operator build outputs generated when the TF Adapter is called for online inference on the NPU.

- result: quantization result folder, including the quantized model file and its quantization information file MobileNetV2_quant.json (named after the quantized model).

- MobileNetV2_ascend_quant.json: quantization information file. This file gives the node mapping between the quantized model and the original model and is used for accuracy comparison between the quantized model and the original model.

- MobileNetV2_ascend_quantized.pb: quantized model that can serve for accuracy simulation in the TensorFlow environment and be deployed on the Ascend AI Processor.

When a model is re-quantized, the existing result files will be overwritten.

4. (Optional) Convert the quantized model into an offline model adapted to the Ascend AI Processor.
   a. Set up the inference environment by referring to CANN Software Installation Guide of Ascend 310 AI Processor.
   b. Upload the quantized model to the environment.
   c. Convert the model by referring to ATC Instructions of Ascend 310 AI Processor.

9.3.2 Quantizing an Object Detection Network

Prerequisites

- Model
  Upload the TensorFlow model to be quantized to any directory on the Linux server as the AMCT installation user. This section uses the yolov3/pre_model/yolov3_coco.pb model available in the sample package as an example.
If you choose to use your own model, you are advised to perform inference in the TensorFlow environment in advance to test if it can run properly on NPU in the TensorFlow environment with expected accuracy.

- **Image dataset**
  After the model is quantized using AMCT, perform inference with the model to test the model accuracy. Use the dataset that matches the model.

  Upload the dataset matching the model to any directory on the Linux server as the AMCT installation user. This section uses the `yolov3/image/detection.jpg` dataset available in the sample package as an example.

- **Calibration dataset**
  The calibration dataset is used to generate the quantization factors to guarantee the accuracy.

  The process of calculating the quantization factors is referred to as calibration. Perform inference using the quantized network with one or more batches of a subset of images from the validation dataset to complete calibration.

  Upload the calibration dataset file to any directory on the Linux server as the AMCT installation user. This section uses the `yolov3/image/calibration.jpg` calibration dataset available in the sample package as an example.

**Procedure**

1. Go to the `ascend_sample/yolov3` directory and run the following command to quantize the YOLOv3 network:

   ```
   python3.7.5 yolov3_calibration.py
   ```

   If messages similar to the following are displayed, the quantization is successful:

   ```
   INFO - [AMCT][save_model]: The model is saved in $HOME/amct/amct_tf/ascend_sample/yolov3/
   result/YOLOv3_ascend_quantized.pb     //Directory of the quantized model
   origin.png save successfully!      //Precheck result
   quantize.png save successfully!    //Postcheck result
   ```

2. After the quantization is successful, find the following subdirectories generated in the directory of the quantized model:

   - **calibration_tmp**:
     - `config_ascend.json`: configuration file that describes how to quantize each layer in the model. If a quantization configuration file already exists in the directory of the quantization script, the existing quantization configuration file is overwritten by a new one with the same name in a call to 9.6.2.1 create_quant_config_ascend. If not that case, a new quantization configuration file is created.

     If the accuracy of model inference drops significantly after quantization, you can modify the `config_ascend.json` file by referring to 9.3.3 Quantization Configuration.

     - `origin.png`: precheck result.

     - `quantize.png`: postcheck result.
- **record_ascend.txt**: quantization factor record file. For details about the prototype definition of the file, see 9.7.3 Quantization Factor Record File.
  - **amct_log**: AMCT log folder, including the quantization log `amct_tensorflow.log`.
  - **tmp**:
    - **fusion_result.json**: file generated when the TF Adapter is called for online inference on the NPU, which stores the graph fusion and UB fusion patterns.
    - **check_result.tf.json**: file generated when the TF Adapter is called for online inference on the NPU, which stores the operators that cannot run on the NPU in the inference graph.
    - **kernel_meta**: directory of operator build outputs generated when the TF Adapter is called for online inference on the NPU.
  - **result**: quantization result folder, containing the quantized model file and the quantization information file.
    - **YOLOv3_ascend_quant.json**: quantization information file (named after the quantized model). This file gives the node mapping between the quantized model and the original model and is used for accuracy comparison between the quantized model and the original model.
    - **YOLOv3_ascend_quantized.pb**: quantized model that can serve for accuracy simulation in the TensorFlow environment and be deployed on the Ascend AI Processor.

When a model is re-quantized, the existing result files will be overwritten.

3. (Optional) Convert the quantized model into an offline model adapted to the Ascend AI Processor.
   a. Set up the inference environment by referring to CANN Software Installation Guide of Ascend 310 AI Processor.
   b. Upload the quantized model to the environment.
   c. Convert the model by referring to ATC Instructions of Ascend 310 AI Processor.

### 9.3.3 Quantization Configuration

This section describes the quantization configuration file of image classification networks.

#### 9.3.3.1 Overview

If the inference accuracy of the `config_ascend.json` quantization configuration file generated by the 9.6.2.1 `create_quant_config_ascend` call does not meet the requirements, you need to tune the `config_ascend.json` file until the accuracy is as expected. The following is an example of the file content.

```json
{
  "version":1,
  "activation_offset":true,
  ...
}
```
9.3.3.2 Configuration File Options

The following tables describe the parameters in the configuration file.

Table 9-7 version

<table>
<thead>
<tr>
<th>Function</th>
<th>Version number of the quantization configuration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
</tr>
<tr>
<td>Value Range</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>Currently, only version 1 is available.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>1</td>
</tr>
</tbody>
</table>
### Table 9-8 activation_offset

<table>
<thead>
<tr>
<th>Function</th>
<th>Symmetric quantization or asymmetric quantization select for activation quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td><strong>true</strong> or <strong>false</strong></td>
</tr>
<tr>
<td>Description</td>
<td>If it is set to <strong>true</strong>, asymmetric quantization is used. If it is set to <strong>false</strong>, symmetric quantization is used.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>true</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 9-9 do_fusion

<table>
<thead>
<tr>
<th>Function</th>
<th>Fusion switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Value Range</td>
<td><strong>true</strong> or <strong>false</strong></td>
</tr>
<tr>
<td>Description</td>
<td>If it is set to <strong>true</strong>, fusion is enabled. If it is set to <strong>false</strong>, fusion is disabled. For details about layers that support fusion and fusion patterns, see 9.1.3 Fusion Support.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>true</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 9-10 skip_fusion_layers

<table>
<thead>
<tr>
<th>Function</th>
<th>Layer skip in BN fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value Range</td>
<td>Layers that support fusion. For details about layers that support fusion and fusion patterns, see 9.1.3 Fusion Support.</td>
</tr>
</tbody>
</table>

---

CANN
Development Auxiliary Tool Guide (Training) 9 AMCT Instructions (TensorFlow, Ascend 910)

Issue 01 (2021-07-01) Copyright © Huawei Technologies Co., Ltd. 438
<table>
<thead>
<tr>
<th>Description</th>
<th>Layers to skip in BN fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recommended Value</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 9-11 layer_config**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization configuration of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>object</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Includes the following parameters:</td>
</tr>
<tr>
<td></td>
<td>● quant_enable</td>
</tr>
<tr>
<td></td>
<td>● activation_quant_params</td>
</tr>
<tr>
<td></td>
<td>● weight_quant_params</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 9-12 quant_enable**

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization enable by layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Value Range</strong></td>
<td>true or false</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If it is set to true, the layer is to be quantized. If it is set to false, otherwise.</td>
</tr>
<tr>
<td><strong>Recommended Value</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>Required/Optional</strong></td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 9-13 activation_quant_params**

<table>
<thead>
<tr>
<th>Function</th>
<th>Activation quantization parameters of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>object</td>
</tr>
</tbody>
</table>
### Table 9-14 weight_quant_params

<table>
<thead>
<tr>
<th>Function</th>
<th>Weight quantization parameters of a layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>object</td>
</tr>
<tr>
<td>Value Range</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>Includes the following parameters:</td>
</tr>
<tr>
<td></td>
<td>• max_percentile</td>
</tr>
<tr>
<td></td>
<td>• min_percentile</td>
</tr>
<tr>
<td></td>
<td>• search_range</td>
</tr>
<tr>
<td></td>
<td>• search_step</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>None</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 9-15 max_percentile

<table>
<thead>
<tr>
<th>Function</th>
<th>Upper search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0.5, 1]</td>
</tr>
<tr>
<td>Description</td>
<td>Indicates the maximum number to be considered as the search result among a group of numbers in descending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the maximum, that is, the largest number. A larger value indicates that the upper clip limit is closer to the maximum value of the data to be quantized.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>0.999999</td>
</tr>
</tbody>
</table>
### Table 9-16 min_percentile

<table>
<thead>
<tr>
<th>Function</th>
<th>Lower search limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value Range</td>
<td>(0.5, 1]</td>
</tr>
<tr>
<td>Description</td>
<td>Indicates the minimum number to be considered as the search result among a group of numbers in ascending order. For example, if there are 100 numbers, the value 1.0 indicates that number 0 (100 – 100 x 1.0) is considered as the minimum, that is, the smallest number. A larger value indicates that the lower clip limit is closer to the minimum value of the data to be quantized.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>0.999999</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 9-17 search_range

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search range: [search_range_start, search_range_end]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>A list of two floats</td>
</tr>
<tr>
<td>Value Range</td>
<td>0 &lt; search_range_start &lt; search_range_end</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the quantization factor search range:</td>
</tr>
<tr>
<td></td>
<td>• search_range_start: search start.</td>
</tr>
<tr>
<td></td>
<td>• search_range_end: search end.</td>
</tr>
<tr>
<td>Recommended Value</td>
<td>[0.7, 1.3]</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Table 9-18 search_step

<table>
<thead>
<tr>
<th>Function</th>
<th>Quantization factor search step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>float</td>
</tr>
</tbody>
</table>
### 9.3.3.3 Configuration Tuning

If the inference accuracy of the model quantized based on the default configuration in the config_ascend.json file drops significantly, perform the following steps to tune the quantization configuration file:

**Step 1** Execute the quantization script in the amct_tensorflow_sample.tar.gz package to perform quantization based on the default configuration generated by the 9.6.2.1 create_quant_config_ascend call.

**Step 2** If the accuracy of the model quantized based on the default configuration is as expected, configuration tuning ends. Otherwise, go to the next step.

**Step 3** Tune quant_enable in the quantization configuration file.

quant_enable specifies whether to quantize a layer. If it is set to true, the layer is to be quantized. If it is set to false, otherwise. If the configuration of a layer is not present, the quantization of the layer is skipped. Generally, specifying fewer layers to quantize improves quantization accuracy. When the network accuracy is not as expected, locate the quantization-sensitive layers (whose error increases significantly after quantization, such as the top layer, bottom layer, depthwise convolutional layer, and layers with few parameters) in the network, and disable quantization on these layers as needed.
Step 4  If the accuracy of the model quantized in Step 3 is as expected, configuration tuning ends. Otherwise, go to Step 5.

Step 5  Tune the values of activation_quant_params and weight_quant_params in the quantization configuration file.

- Data is clipped to the range $[left, right]$ specified by the activation_quant_params parameters. Generally, values distributed near a boundary are sparse, and clip may be performed on all the values, to improve the accuracy. A larger value of min_percentile (max_percentile) indicates that left (right) is closer to the minimum value (maximum value) of the to-be-quantized data. search_range and search_step affect the range of $[left, right]$. Generally, a larger value of search_range and a smaller value of search_step may achieve higher quantization accuracy, but the quantization takes more time.

- channel_wise in weight_quant_params determines whether to use a different quantization factor for each channel during weight quantization. If it is set to true, channels are separately quantized using different quantization factors. If it is set to false, all channels are quantized altogether using the same quantization factors. Generally, the inference accuracy is higher if the channels are separately quantized. However, the MatMul and AvgPool layers are channel-irrelevant. Therefore, this parameter does not take effect on these layers.

Step 6  If the accuracy of the model quantized in Step 5 is as expected, configuration tuning ends. Otherwise, it indicates that quantization has severe adverse impact on the inference accuracy. In this case, remove the quantization configuration.

----End
9.4 AMCT Update

The latest AMCT release allows you to access to the latest features. Before updating AMCT, uninstall the existing installation by referring to 9.5 AMCT Uninstallation, and then install the latest version by referring to 9.2 AMCT Installation.

9.5 AMCT Uninstallation

You can uninstall the AMCT as follows:
1. Run the following command in any directory on the Linux server as the AMCT installation user:
   ```
   pip3.7.5 uninstall amct_tensorflow_ascend
   ```

2. When the following information is displayed, enter `y`:
   ```
   Uninstalling amct-tensorflow-ascend-{version}:
   Would remove:
   ...
   Proceed (y/n)? y
   ```
   If a message similar to the following is displayed, the uninstallation is successful:
   ```
   Successfully uninstalled amct-tensorflow-ascend-{version}
   ```
   The installed TensorFlow will not be uninstalled during the uninstallation.

9.6 API Description

9.6.1 Common APIs

9.6.1.1 set_logging_level

**Description**

Sets the logging levels of the log messages printed to the screen and those saved to the `amct_log/amct_tensorflow.log` file.

**Prototype**

```python
set_logging_level(print_level='info', save_level='info')
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_level</td>
<td>Input</td>
<td>Logging level of the log messages printed to the screen.</td>
<td>Default: <code>info</code> A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>debug</code>: DEBUG, INFO, WARNING, and ERROR logs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>info</code>: INFO, WARNING, and ERROR logs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>warning</code>: WARNING and ERROR logs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>error</code>: ERROR logs.</td>
<td></td>
</tr>
</tbody>
</table>
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>save_level</td>
<td>Input</td>
<td>Logging level of log messages saved to the <code>quant_info.log</code> file.</td>
<td>Default: <strong>info</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>debug</strong>: DEBUG, INFO, WARNING, and ERROR logs.</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>info</strong>: INFO, WARNING, and ERROR logs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>warning</strong>: WARNING and ERROR logs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>error</strong>: ERROR logs.</td>
<td></td>
</tr>
</tbody>
</table>

### Logging Level Table

<table>
<thead>
<tr>
<th>Logging Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG</td>
<td>Detailed quantization processing information, including the quantization factors (scale and offset) and related debugging information.</td>
</tr>
<tr>
<td>INFO</td>
<td>Brief quantization processing information, including the quantized layer names and BN fusion information.</td>
</tr>
<tr>
<td>WARNING</td>
<td>Warning messages during quantization.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Error messages during quantization.</td>
</tr>
</tbody>
</table>

The logging level is case insensitive. That is, **Info**, **info**, and **INFO** are all valid values.

### Returns

None

### Example

```python
import amct_tensorflow as amct
amct.set_logging_level(print_level="info", save_level="info")
amct.quantize_model(
    graph=tf.get_default_graph(),
    config_file="./configs/config.json",
    record_file="./record_scale_offset.txt")
```

### 9.6.2 Post-training Quantization
9.6.2.1 create_quant_config_ascend

**Description**

Applies to post-training quantization. Finds all quantization-capable layers in a graph, creates a quantization configuration file, and writes the quantization configuration information of the quantization-capable layers to the file.

**Prototype**

```python
create_quant_config_ascend(config_file, graph, skip_layers=None, activation_offset=True, config_definition=None)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the quantization configuration file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The existing file (if available) in the directory will be overwritten upon this API call.</td>
<td></td>
</tr>
<tr>
<td>graph</td>
<td>Input</td>
<td>A <code>tf.Graph</code> of the model to be quantized.</td>
<td>A <code>tf.Graph</code>.</td>
</tr>
<tr>
<td>skip_layers</td>
<td>Input</td>
<td>Quantizable layers in the <code>tf.Graph</code> to skip.</td>
<td>Default: <strong>None</strong>&lt;br&gt;A list of strings.&lt;br&gt;Restriction: If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| activation_offset | Input        | Whether to quantize activations with offset.                               | Default: **True**  
A bool.  
Restriction: If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect. |
| config_definition | Input        | Whether to create a simplified quantization configuration file quant.cfg from the calibration_config_ascend_tf.proto file in /amct_tensorflow/proto/calibration_config_ascend_tf.proto in the AMCT installation path.  
For details about the parameters in the calibration_config_ascend_tf.proto template and the generated simplified quantization configuration file quant.cfg, see 9.7.2.1 Simplified Configuration File Template (calibration_config_ascend_tf.proto). | Default: **None**  
A string.  
Restriction: If it is set to None, a configuration file is generated based on the residual arguments (skip_layers and activation_offset). Otherwise, a configuration file in JSON format is generated based on this argument. |

**Returns**

None

**Outputs**

Outputs a quantization configuration file in JSON format. (When quantization is performed again, the existing one output by this API call will be overwritten.)

Assume that only layer_name1 and layer_name2 in a graph support quantization. The quantization configuration file generated by the 9.6.2.1 create_quant_config_ascend call is as follows:

```
{
    "version":1,
```
"activation_offset":true,
"do_fusion":true,
"skip_fusion_layers":[],
"MobilenetV2/Conv/Conv2D":{
  "quant_enable":true,
  "activation_quant_params":{
    "max_percentile":0.999999,
    "min_percentile":0.999999,
    "search_range":[
      0.7,
      1.3
    ],
    "search_step":0.01
  },
  "weight_quant_params":{
    "channel_wise":true
  }
},
"MobilenetV2/Conv_1/Conv2D":{
  "quant_enable":true,
  "activation_quant_params":{
    "max_percentile":0.999999,
    "min_percentile":0.999999,
    "search_range":[
      0.7,
      1.3
    ],
    "search_step":0.01
  },
  "weight_quant_params":{
    "channel_wise":true
  }
},
"MobilenetV2/Logits/AvgPool":{
  "quant_enable":true,
  "activation_quant_params":{
    "max_percentile":0.999999,
    "min_percentile":0.999999,
    "search_range":[
      0.7,
      1.3
    ],
    "search_step":0.01
  },
  "weight_quant_params":{
    "channel_wise":false
  }
}

Example

import amct_tensorflow as amct
# Create a graph of the network to be quantized.
network = build_network()
# Create a quantization configuration file.
amct.create_quant_config_ascend(config_file=":/configs/config.json",
  graph=tf.get_default_graph(),
  skip_layers=None,
  activation_offset=True)

9.6.2.2 quantize_model_ascend

Description

Applies to post-training quantization. Quantizes a graph based on the
quantization configuration file, inserts the quantization operators, generates a
quantization factor record file record_file, and returns the list of newly added operators.

Prototype

calibration_graph, calibration_outputs=quantize_model_ascend(graph, outputs, config_file, record_file)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph</td>
<td>Input</td>
<td>A tf.Graph of the model to be quantized.</td>
<td>A tf.Graph. Must be an inference graph containing no training-mode operators. For example, is_training of the FusedBatchNormV3 operator must be False. The graph is loaded with trained weights.</td>
</tr>
<tr>
<td>outputs</td>
<td>Input</td>
<td>List of output operators of the graph.</td>
<td>A list of strings.</td>
</tr>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization configuration file generated by the user, which is used to specify the configuration of the quantization layer in the tf.Graph graph of the model</td>
<td>A string.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td>calibration_graph</td>
<td>Return</td>
<td>Result graph, with quantization operators inserted.</td>
<td>A tf.Graph.</td>
</tr>
<tr>
<td>calibration_outputs</td>
<td>Return</td>
<td>Output operators of calibration_graph</td>
<td>A list of strings.</td>
</tr>
</tbody>
</table>

Returns

A list of quantized layers on the network.

9.6.2.2 quantize_model_ascend performs BN fusion on the graph. If the outputs of the network model contain the BN layer and the BN layer is also converged, the
output node of the network changes. For example, Conv+BN (or Conv+BiasAdd +BN) is fused into Conv+BiasAdd, and an output node equivalent to BN is a BiasAdd node.

Example

```python
import amct_tensorflow as amct
# Create a network to be quantized.
network = build_network()

# Quantize the model.
calibration_graph, calibration_outputs = amct.quantize_model_ascend(
    graph=tf.get_default_graph(),
    config_file='./configs/config.json',
    record_file='./record_scale_offset.txt')
```

9.6.2.3 save_model_ascend

Description

Applies to post-training quantization. Saves the original .pb model file to be quantized as a .pb model file that can be used for both accuracy simulation in the NPU environment and inference on the Ascend AI Processor based on the record_file quantization factor record file.

Restrictions

- The model file passed to the API call must be in .pb format. You might need to convert your models to the .pb format in advance.
- The quantization factor record file passed to the API call is generated in the 9.6.2.2 quantize_model_ascend phase. The factor values will be filled in the model inference phase.

Prototype

```python
save_model_ascend(pb_model, outputs, record_file, save_path)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb_model</td>
<td>Input</td>
<td>Original .pb model file to be quantized. A string. Must be an inference graph containing no training-mode operators. For example, is_training of the FusedBatchNormV3 operator must be False.</td>
<td></td>
</tr>
<tr>
<td>outputs</td>
<td>Input</td>
<td>List of output operators of the graph. A list of strings.</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generate a quantized model file based on the file, quantization configuration file, and original .pb model file.</td>
<td></td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path.</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Must be prefixed with the model name, for example, ./quantized_model/*model.</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

None

**Outputs**

Outputs a .pb model file that can be used for both accuracy simulation on NPU and inference on the Ascend AI Processor.

When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.

**Example**

```python
import amct_tensorflow as amct
# Perform network inference and complete quantization during the inference.
with calibration_graph.as_default():
    sess = tf.session(prepare_config("npu"))
    sess.run(calibration_outputs, feed_dict={inputs: inputs_data})
# Insert the API call and save the quantized model as a .pb file.
amct.save_model_ascend(pb_model="./user_model.pb",
    outputs=['model/outputs'],
    record_file="./record_scale_offset.txt",
    save_path="./inference/model")
```

**9.6.2.4 create_quant_config**

**Description**

Applies to post-training quantization. Finds all quantization-capable layers in a graph, creates a quantization configuration file, and writes the quantization configuration information of the quantization-capable layers to the file.

**Prototype**

```python
create_quant_config(config_file, graph, skip_layers=None, batch_num=1,
activation_offset=True, config_defination=None)
```
# Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Directory of the quantization configuration file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The existing file (if available) in the directory will be overwritten upon</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>this API call.</td>
<td></td>
</tr>
<tr>
<td>graph</td>
<td>Input</td>
<td>A <code>tf.Graph</code> of the model to be quantized.</td>
<td>A <code>tf.Graph</code>.</td>
</tr>
<tr>
<td>skip_layers</td>
<td>Input</td>
<td>Quantizable layers in the <code>tf.Graph</code> to skip.</td>
<td>Default: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A list of strings.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Restriction: If a simplified quantization configuration file is used as the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>input, this parameter must be set in the configuration file. In this case,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>the parameter setting in the input does not take effect.</td>
<td></td>
</tr>
<tr>
<td>batch_num</td>
<td>Input</td>
<td>Number of batches taken to generate the quantization factors.</td>
<td>An int.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: any integer larger than 0.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Restrictions:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>batch_num</code> cannot be too large. The product of <code>batch_num</code> and <code>batch_size</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>equals to the number of images used during quantization. Too many images</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>consume too much memory.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If a simplified quantization configuration file is used as the input, this</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameter must be set in the configuration file. In this case, the parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>setting in the input does not take effect.</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Return</td>
<td>Description</td>
<td>Restriction</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| activation_offset  | Input        | Whether to quantize activations with offset.                                | Default: true
|                    |              |                                                                             | A bool.                                                                     |
|                    |              |                                                                             | Restriction: If a simplified quantization configuration file is used as the input, this parameter must be set in the configuration file. In this case, the parameter setting in the input does not take effect. |
| config_definition  | Input        | Whether to create a simplified quantization configuration file quantcfg from the calibration_config_tf.proto file in /amct_tensorflow/proto/calibration_config_tf.proto in the AMCT installation path. For details about the parameters in the calibration_config_tf.proto template and the generated simplified quantization configuration file quantcfg, see 9.7.2.2 Simplified Configuration File Template (calibration_config_tf.proto). | Default: None
|                    |              |                                                                             | A string.                                                                   |
|                    |              |                                                                             | Restriction: If it is set to None, a configuration file is generated based on the residual arguments (skip_layers, batch_num, and activation_offset). Otherwise, a configuration file in JSON format is generated based on this argument. |

**Returns**

None

**Outputs**

Outputs a quantization configuration file in JSON format. (When quantization is performed again, the existing one output by this API call will be overwritten.)

```json
{
    "version": 1,
    "batch_num": 1,
    "activation_offset": true,
    "joint_quant": false,
    "do_fusion": true,
    "skip_fusion_layers": [],
    "MobilenetV2/Conv/Conv2D": {
        "quant_enable": true,
        "activation_quant_params": {
```
"max_percentile":0.999999,
"min_percentile":0.999999,
"search_range":[
  0.7,
  1.3
],
"search_step":0.01
},
"weight_quant_params":{
  "wts_algo":"arq_quantize",
  "channel_wise":true
},
"MobilenetV2/Conv_1/Conv2D":{
  "quant_enable":true,
  "activation_quant_params":{
    "max_percentile":0.999999,
    "min_percentile":0.999999,
    "search_range":[
      0.7,
      1.3
    ],
    "search_step":0.01
  },
  "weight_quant_params":{
    "wts_algo":"arq_quantize",
    "channel_wise":true
  }
},
"MobilenetV2/Logits/AvgPool":{
  "quant_enable":true,
  "activation_quant_params":{
    "max_percentile":0.999999,
    "min_percentile":0.999999,
    "search_range":[
      0.7,
      1.3
    ],
    "search_step":0.01
  },
  "weight_quant_params":{
    "wts_algo":"arq_quantize",
    "channel_wise":false
  }
}

Example

import amct_tensorflow as amct
# Create a graph of the network to be quantized.
network = build_network()
# Create a quantization configuration file.
amct.create_quant_config(config_file="./configs/config.json",
  graph=tf.get_default_graph(),
  skip_layers=None,
  batch_num=1,
  activation_offset=True)

9.6.2.5 quantize_model

Description

Applies to post-training quantization. Quantizes a graph based on the quantization configuration file, inserts the quantization operators, generates a quantization factor record file `record_file`, and returns the list of newly added operators.
Prototype

\[
\text{quant_add_ops} = \text{quantize_model}(\text{graph}, \text{config_file}, \text{record_file})
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph</td>
<td>Input</td>
<td>A tf.Graph of the model to be quantized.</td>
<td>A tf.Graph. Must be an inference graph containing no training-mode operators. For example, is_training of the FusedBatchNormV3 operator must be False.</td>
</tr>
<tr>
<td>config_file</td>
<td>Input</td>
<td>Quantization configuration file generated by the user, which is used to specify the configuration of the quantization layer in the tf.Graph graph of the model</td>
<td>A string.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name.</td>
<td>A string.</td>
</tr>
<tr>
<td>quant_add_ops</td>
<td>Return</td>
<td>List of operators to be inserted for quantization.</td>
<td>A list of tf.Variables.</td>
</tr>
</tbody>
</table>

Returns

A list of quantized layers on the network.

The 9.6.2.5 quantize\_model call performs fusion on the graph, which might alter the output nodes. For example, Conv+BN (or Conv+BiasAdd+BN) is fused into Conv+BiasAdd, and an output node equivalent to BN is a BiasAdd node.

Example

```python
import amct_tensorflow as amct
# Create a network to be quantized.
network = build_network()

# Quantize the model.
amct.quantize_model(
    graph=tf.get_default_graph(),
    config_file="./configs/config.json",
    record_file="./record_scale_offset.txt")
```
9.6.2.6 save_model

Description

Applies to post-training quantization. Saves the original .pb model file to be quantized as a .pb model file that can be used for both accuracy simulation in the TensorFlow environment and inference on the Ascend AI Processor based on the record_file quantization factor record file.

Restrictions

- This API is called after batch_num is reached. Otherwise, the quantization factors are incorrect and the quantization result is compromised.
- The model file passed to the API call must be in .pb format. You might need to convert your models to the .pb format in advance.
- The quantization factor record file passed to the API call is generated in the 9.6.2.5 quantize_model phase. The factor values will be filled in the model inference phase.

Prototype

save_model(pb_model, outputs, record_file, save_path)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb_model</td>
<td>Input</td>
<td>Original .pb model file to be quantized.</td>
<td>A string. Must be an inference graph containing no training-mode operators. For example, is_training of the FusedBatchNormV3 operator must be False.</td>
</tr>
<tr>
<td>outputs</td>
<td>Input</td>
<td>List of output operators of the graph.</td>
<td>A list of strings.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Directory of the quantization factor record file, including the file name. Generate a quantized model file based on the file, quantization configuration file, and original .pb model file.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, ./quantized_model/*model.</td>
<td>A string.</td>
</tr>
</tbody>
</table>
Returns

None

Outputs

Outputs a .pb model file that can be used for both accuracy simulation in the TensorFlow environment and inference on the Ascend AI Processor.

When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.

Example

```python
import amct_tensorflow as amct

# Perform network inference and complete quantization during the inference.
for i in batch_num:
    sess.run(outputs, feed_dict={inputs: inputs_data})
# Insert the API call and save the quantized model as a .pb file.
amct.save_model(pb_model="./user_model.pb",
                outputs=["model/outputs"],
                record_file="./record_scale_offset.txt",
                save_path="./inference/model")
```

9.6.2.7 convert_model

Description

Based on the computed quantization factors, converts a TensorFlow model to a model for both accuracy simulation in the TensorFlow environment and inference on the Ascend AI Processor.

Restrictions

- The user model must match the quantization factor record file. For example, if the "Conv+BN" composite is fused before computing the quantization factors of Conv, the "Conv+BN" composite in the TensorFlow model to be converted also needs to be fused in advance.
- The format and content of the quantization factor record file must comply with the AMCT requirements defined in 5.11.4 Quantization Factor Record File.
- The quantization-capable layers include: Conv2D, MatMul, DepthwiseConv2dNative (dilation = 1), Conv2DBackpropInput (dilation = 1), and AvgPool.
- This API supports the fusion of the "Conv+BN", "Depthwise_Conv+BN", and "Group_conv+BN" composites in the user model. Layer-level fusion switch is supported.
- Quantization of only an original floating-point model is supported. The model cannot be quantized if the input model contains any of the following custom quantization layers: QuantIfmr, QuantArq, SearchN, AscendQuant, AscendDequant, AscendAntiQuant, and AscendWeightQuant.

Prototype

```
convert_model(pb_model, outputs, record_file, save_path)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Return</th>
<th>Description</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb_model</td>
<td>Input</td>
<td>Original .pb model file to be quantized.</td>
<td>A string. Must be an inference graph containing no training-mode operators. For example, <code>is_training</code> of the FusedBatchNormV3 operator must be <code>False</code>.</td>
</tr>
<tr>
<td>outputs</td>
<td>Input</td>
<td>List of output operators of the graph.</td>
<td>A list.</td>
</tr>
<tr>
<td>record_file</td>
<td>Input</td>
<td>Path of the quantization factor record file (.txt) computed by the user.</td>
<td>A string.</td>
</tr>
<tr>
<td>save_path</td>
<td>Input</td>
<td>Model save path. Must be prefixed with the model name, for example, <code>./quantized_model/*model*</code>.</td>
<td>A string.</td>
</tr>
</tbody>
</table>

Returns

None

Outputs

Outputs a .pb model file that can be used for both accuracy simulation in the TensorFlow environment and inference on the Ascend AI Processor.

When quantization is performed again, the existing files in the output directory will be overwritten upon this API call.

Example

```python
import amct_tensorflow as amct
convert_model(pb_model='./user_model.pb',
              outputs=['model/outputs'],
              record_file='./record_quantized.txt',
              save_path='./quantized_model/model')
```

9.7 Appendixes
9.7.1 Sample Package Directory Structure

Table 9-20 Directory structure of the sample package

<table>
<thead>
<tr>
<th>Level-1 Directory</th>
<th>Level-2 Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mobilenetv2</td>
<td>-</td>
<td>Image classification directory.</td>
</tr>
<tr>
<td></td>
<td>image</td>
<td>- classification.jpg: dataset used by the image classification network for inference.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Other .jpg files: calibration dataset used by the image classification network, including 32 images.</td>
</tr>
<tr>
<td></td>
<td>image_reference.txt</td>
<td>Link to images in the calibration dataset.</td>
</tr>
<tr>
<td></td>
<td>mobilenetv2_calibration.py</td>
<td>Quantization script.</td>
</tr>
<tr>
<td></td>
<td>pre_model/mobilenet_v2.pb</td>
<td>Original MobileNetV2 model file.</td>
</tr>
<tr>
<td>yolov3</td>
<td>-</td>
<td>Object detection directory.</td>
</tr>
<tr>
<td></td>
<td>image</td>
<td>- calibration.jpg: calibration image used by the object detection network.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- COCO_labels.txt: label dictionary used by the object detection network.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- detection.jpg: dataset used by the object detection network for inference.</td>
</tr>
<tr>
<td></td>
<td>pre_model/yolov3_coco.pb</td>
<td>Model file.</td>
</tr>
<tr>
<td></td>
<td>yolo_quant_ascend.cfg</td>
<td>Simplified quantization configuration file.</td>
</tr>
<tr>
<td></td>
<td>yolov3_calibration.py</td>
<td>Quantization script.</td>
</tr>
</tbody>
</table>

9.7.2 Simplified Configuration File

9.7.2.1 Simplified Configuration File Template (calibration_config_ascend_tf.proto)

Table 9-22 describes the parameters in the calibration_config_ascend_tf.proto template.
Table 9-21 Parameter description

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified post-training quantization configuration file of AMCT.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>activation_offset</td>
<td>Whether to quantize activations with offset.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_layers</td>
<td>Layers to skip during quantization.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_layer_types</td>
<td>Types of layers to skip during quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td></td>
<td>common_config</td>
<td>Common quantization configuration. If a layer is not overridden by override_layer_types or override_layer_configs, this configuration is used.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>Override LayerType</td>
<td>override_layer_types</td>
<td>Type of layers to override.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>Override Layer</td>
<td>override_layer_configs</td>
<td>Layer to override.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>do_fusion</td>
<td>BN fusion switch. Defaults to true.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
<td>--------</td>
<td>------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>string</td>
<td>skip_fusion_layers</td>
<td>Layers to skip in BN fusion.</td>
</tr>
<tr>
<td>OverrideLayerType</td>
<td>Required</td>
<td>string</td>
<td>layer_type</td>
<td>Type of layers to quantize.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>CalibratonConfig</td>
<td>calibration_config</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td>OverrideLayer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization configuration overriding by layer.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>layer_name</td>
<td>Layer to override.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>CalibratonConfig</td>
<td>calibration_config</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td>CalibrationConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Calibration-based quantization configuration.</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>ARQuantize</td>
<td>arq_quantize</td>
<td>Weight quantization algorithm.</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>FMRQuantize</td>
<td>ifmr_quantize</td>
<td>Activation quantization algorithm.</td>
</tr>
<tr>
<td>ARQuantize</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ARQ quantization algorithm configuration.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td>------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FMRQuantize</td>
<td>Optional</td>
<td>bool</td>
<td>channel_wise</td>
<td>Whether to use different quantization factors for each channel.</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>float</td>
<td>search_range_start</td>
<td>FMR quantization algorithm configuration.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>search_range_end</td>
<td>Quantization factor search start.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>search_step</td>
<td>Quantization factor search step.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>max_percentile</td>
<td>Upper search limit.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>min_percentile</td>
<td>Lower search limit.</td>
</tr>
</tbody>
</table>

The following is an example simplified configuration file (*quant.cfg*).

```yaml
# Global quantization parameters
activation_offset : true
skip_layers : "conv_1"
skip_layer_types:"Conv2D"
do_fusion: true
skip_fusion_layers : "conv_1"
common_config : {
    arq_quantize : {
        channel_wise : true
    }
}
ifmr_quantize : {
    search_range_start : 0.7
    search_range_end : 1.3
    search_step : 0.01
    max_percentile : 0.999999
    min_percentile : 0.999999
}
```
override_layer_types : {
  layer_type : "MatMul"
  calibration_config : {
    arq_quantize : {
      channel_wise : false
    }
    ifmr_quantize : {
      search_range_start : 0.8
      search_range_end : 1.2
      search_step : 0.02
      max_percentile : 0.999999
      min_percentile : 0.999999
    }
  }
}
override_layer_configs : {
  layer_name : "conv_2"
  calibration_config : {
    arq_quantize : {
      channel_wise : true
    }
    ifmr_quantize : {
      search_range_start : 0.8
      search_range_end : 1.2
      search_step : 0.02
      max_percentile : 0.999999
      min_percentile : 0.999999
    }
  }
}

9.7.2.2 Simplified Configuration File Template (calibration_config_tf.proto)

Table 9-22 describes the parameters in the calibration_config_tf.proto template.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMCTConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Simplified post-training quantization configuration file of AMCT.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>uint32</td>
<td>batch_num</td>
<td>Batch count used for quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>bool</td>
<td>activation_offset</td>
<td>Whether to quantize activations with offset.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td>-----------</td>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>joint_quant</td>
<td>Eltwise joint quantization switch. Defaults to <code>false</code>, indicating that joint quantization is disabled. If it is set to <code>true</code>, the network performance may improve but the accuracy may be compromised.</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>string</td>
<td>skip_layers</td>
<td>Layers to skip during quantization.</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>string</td>
<td>skip_layer_types</td>
<td>Types of layers to skip during quantization.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>NuqConfig</td>
<td>nuq_config</td>
<td>Non-uniform quantization configuration. This version does not support non-uniform quantization.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>CalibratorConfig</td>
<td>common_config</td>
<td>Common quantization configuration. If a layer is not overridden by <code>override_layer_types</code> or <code>override_layer_configs</code>, this configuration is used.</td>
<td></td>
</tr>
<tr>
<td>Repeated</td>
<td>OverrideLayerType</td>
<td>override_layer_types</td>
<td>Type of layers to override.</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------</td>
<td>---------------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>Override Layer</td>
<td>override_layer_configs</td>
<td>Layer to override.</td>
</tr>
<tr>
<td>NuqConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Non-uniform quantization configuration. This version does not support non-uniform quantization.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>string</td>
<td>mapping_file</td>
<td>JSON file converted from the offline model converted using the ATC tool from the deployable model generated form uniform quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>NUQuantize</td>
<td>nuq_quantize</td>
<td>Non-uniform quantization configuration.</td>
</tr>
<tr>
<td>OverrideLayerType</td>
<td>Required</td>
<td>string</td>
<td>layer_type</td>
<td>Type of layers to quantize.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>CalibrationConfig</td>
<td>calibration_config</td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td>OverrideLayer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization configuration overriding by layer.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
<td>-------</td>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Requred</td>
<td>string</td>
<td>layer_name</td>
<td></td>
<td>Layer to override.</td>
</tr>
<tr>
<td>Required calibration_config</td>
<td>calibration_config</td>
<td></td>
<td></td>
<td>Quantization configuration to apply.</td>
</tr>
<tr>
<td>CalibrationConfig</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Calibration-based quantization configuration.</td>
</tr>
<tr>
<td>-</td>
<td>ARQuantize</td>
<td>arq_quantize</td>
<td></td>
<td>Weight quantization algorithm.</td>
</tr>
<tr>
<td>-</td>
<td>NUQuantize</td>
<td>nuq_quantize</td>
<td></td>
<td>Weight quantization algorithm.</td>
</tr>
<tr>
<td>-</td>
<td>FMRQuantize</td>
<td>ifmr_quantize</td>
<td></td>
<td>Activation quantization algorithm.</td>
</tr>
<tr>
<td>ARQuantize</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ARQ quantization algorithm configuration.</td>
</tr>
<tr>
<td>Optional</td>
<td>bool</td>
<td>channel_wise</td>
<td></td>
<td>Whether to use different quantization factors for each channel.</td>
</tr>
<tr>
<td>FMRQuantize</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>FMR quantization algorithm configuration.</td>
</tr>
<tr>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>--------</td>
<td>----------------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>float</td>
<td>search_range_start</td>
<td>Quantization factor search start.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>float</td>
<td>search_range_end</td>
<td>Quantization factor search end.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>float</td>
<td>search_step</td>
<td>Quantization factor search step.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>float</td>
<td>max_percentile</td>
<td>Upper search limit.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>float</td>
<td>min_percentile</td>
<td>Lower search limit.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>uint32</td>
<td>num_steps</td>
<td>Non-uniform quantization configuration. This version does not support non-uniform quantization.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>uint32</td>
<td>num_of_iteration</td>
<td>Number of iterations for non-uniform quantization optimization.</td>
<td></td>
</tr>
</tbody>
</table>

The following is an example simplified configuration file (quant.cfg).

```
# Global quantization parameters
batch_num : 2
activation_offset : true
joint_quant : false
skip_layers : "conv_1"
skip_layer_types:"Conv2D"
do_fusion: true
skip_fusion_layers : "conv_1"
common_config : {
  arq_quantize : {
```

The following is an example simplified configuration file (quant.cfg).

```
# Global quantization parameters
batch_num : 2
activation_offset : true
joint_quant : false
skip_layers : "conv_1"
skip_layer_types:"Conv2D"
do_fusion: true
skip_fusion_layers : "conv_1"
common_config : {
  arq_quantize : {
```
9.7.3 Quantization Factor Record File

Prototype

The quantization factor record file is a serialized data structure file based on Protobuf. You can generate a quantized model file by using the quantization configuration file, original network model file, and the quantization factor record file. The Protobuf prototype is defined as follows (find the code in the /amct_tensorflow/proto/scale_offset_record_tf.proto file in the AMCT installation directory).

```protobuf
syntax = "proto2";
package AMCTTensorflow;
// this proto is designed for convert_model API
message SingleLayerRecord {
  optional float scale_d = 1;
  optional int32 offset_d = 2;
  repeated float scale_w = 3;
  repeated int32 offset_w = 4;
  // convert_model does not support this field [shift_bit] yet
  repeated uint32 shift_bit = 5;
  optional bool skip_fusion = 9 [default = false];
}
```
message MapFiledEntry {
  optional string key = 1;
  optional SingleLayerRecord value = 2;
}

message ScaleOffsetRecord {
  repeated MapFiledEntry record = 1;
}

The parameters are described as follows.

<table>
<thead>
<tr>
<th>Message</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleLayerRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Quantization factors for quantization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>float</td>
<td>scale_d</td>
<td>Scale factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>int32</td>
<td>offset_d</td>
<td>Offset factor for activation quantization. Only unified activation quantization is supported.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>float</td>
<td>scale_w</td>
<td>Scale factor for weight quantization. Two quantization modes are supported: scalar (quantizing the weight of the current layer in a unified manner) and vector (quantizing the weight of the current layer channel-wise). The channel-wise quantization mode applies only to the Conv2D, DepthwiseConv2dNative, and Conv2DBackpropInput layers.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>int32</td>
<td>offset_w</td>
<td>Offset factor for weight quantization. Similar to scale_w, it also supports the scalar and vector quantization modes and the dimension configuration must be the same as that of scale_w. Currently weight quantization with offset is not supported, and offset_w must be 0.</td>
</tr>
<tr>
<td></td>
<td>Repeated</td>
<td>uint32</td>
<td>shift_bit</td>
<td>Shift factor. Reserved.</td>
</tr>
<tr>
<td>Message</td>
<td>Required/Optional</td>
<td>Type</td>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td>--------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Optiional</td>
<td>bool</td>
<td>skip_fusion</td>
<td>Whether to skip Conv+BN fusion, Depthwise_Conv+BN fusion, Group_conv+BN fusion, and BatchNorm fusion at the current layer. Defaults to false, indicating that fusion of the preceding types is performed.</td>
</tr>
<tr>
<td>ScaleOffsetRecord</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Map structure. To ensure compatibility, the discrete map structure is used.</td>
</tr>
</tbody>
</table>
| Repeated | MapFiledEntry    | record |            | Each records a quantization factor of a quantization layer and consists of two members:  
|          |             |        |            |   - **key**: layer name.  
|          |             |        |            |   - **value**: quantization factor defined by SingleLayerRecord.            |
| MapFiledEntry | Optiional    | string | key        | Layer name.                                                                  |
|          | Optiional      | SingleLayerRecord | value  | Quantization factor configuration.                                          |

Beware that the Protobuf protocol does not report an error for repeated settings of optional fields. Instead, the most recent settings are used.

For general quantization layers, a range of parameters need to be configured, including `scale_d`, `offset_d`, `scale_w`, and `offset_w`. The `scale_w` and `offset_w` parameters are unavailable for AvgPool since the layer has no weight. An example of the quantization factor record file is as follows:

```protobuf
definition record {
  key: "fc4/Tensordot/MatMul"
  value {
    scale_d: 0.0798481479
    offset_d: 1
    scale_w: 0.00297622895
    offset_w: 0
  }
}
definition record {
  key: "depthwise"
  value {
    scale_d: 0.00962011795
    offset_d: 1
    scale_w: 0.00787108205
    scale_w: 0.00787108205
    scale_w: 0.00787108205
  }
}
```
Quantization Factors

The scale (a floating-point number) and offset quantization factors need to be provided for data and weight quantization. AMCT uses a unified quantization data structure. See the following expression.

\[
data_{\text{int8}} = \text{clip}_{\text{int8}} \left( \text{round} \left( \frac{\text{data}_{\text{float}}}{\text{scale}} \right) + \text{offset} \right)
\]

The value ranges are as follows:

- \( \text{scale} \in \left[ \text{FLT\_EPSILON}, \frac{1}{\text{FLT\_EPSILON}} \right] \) \( \text{FLT\_EPSILON} \approx 1.1920929 \times 10^{-7} \)
- \( \text{offset} \in [-128, 127] \)

Quantization algorithms are classified into symmetric quantization algorithms and asymmetric quantization algorithms.

1. Symmetric quantization algorithm:

   The original high-precision data and quantized int8 data are converted into \( \text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int8}} \), where \( \text{scale} \) is a float32. To indicate positive and negative numbers, the signed int8 data type is used for \( \text{data}_{\text{float}} = \text{scale} \times \text{data}_{\text{int8}} \). The following describes how to convert the original data into the int8 format. \( \text{round} \) is a rounding function. The value to be determined by the quantization algorithm is the constant \( \text{scale} \).

   \[
data_{\text{int8}} = \text{round} \left( \frac{\text{data}_{\text{float}}}{\text{scale}} \right)
\]

Quantization of the weights and activations may be summarized as a process of searching for a scale. Because \( \text{data}_{\text{int8}} \) is a signed number, to ensure symmetry of the ranges represented by positive and negative values, an absolute value operation is first performed on all data, so that the range of the to-be-quantized data is changed to \( \text{data}_{\text{int8}} \), and then \( \text{scale} \) is determined. The range of positive int8 values is \([0, 127]\). Therefore, \( \text{scale} \) can be computed as follows:
Therefore, the range of the int8 values is $[-128 \times \text{scale}, 127 \times \text{scale}]$. Data beyond the range $[-128 \times \text{scale}, 127 \times \text{scale}]$ is saturated to a boundary value, and then the quantization operation shown in the formula is performed.

2. Asymmetric quantization algorithm:
The difference from symmetric quantization algorithms lies in the data conversion mode. The \textit{scale} and \textit{offset} constants also need to be determined. $\text{data}_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset})$

The uint8 data is converted through calculation based on the original high-accuracy data, which is shown in the following formula:

$$\text{data}_{\text{uint8}} = \text{round} \left( \frac{\text{data}_{\text{float}}}{\text{scale}} - \text{offset} \right)$$

\textit{scale} is an fp32, $\text{data}_{\text{uint8}}$ is an unsigned int8, and \textit{offset} is an int8. The data range is $[\text{scale} \times \text{offset}, \text{scale} \times (255 + \text{offset})]$. If a value range of the to-be-quantized data is $[\text{data}_{\text{min}}, \text{data}_{\text{max}}]$, \textit{scale} and \textit{offset} are computed as follows:

$$\text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}, \quad \text{scale} = \frac{\text{data}_{\text{max}} - \text{data}_{\text{min}}}{255}$$

Unified quantization data format: By performing simple data conversion of the asymmetric quantization algorithm formula, the quantized data and the symmetric quantization algorithm are of the same type, that is, int. The specific conversion process is as follows:

int8 quantization is used as an example for description. The input original floating-point data is $\text{data}_{\text{float}}$, the original quantized fixed-point number is $\text{data}_{\text{float}}$, the quantization scale is \textit{scale}, and the quantization offset is $\text{data}_{\text{float}}$ (the algorithm requires zero crossing to avoid too much accuracy drop). The calculation principle of quantization is as follows:

$$\text{data}_{\text{float}} = \text{scale} \times (\text{data}_{\text{uint8}} + \text{offset}) = \text{scale} \times (\text{data}_{\text{int8}} + \text{offset} + 128) = \text{scale} \times (\text{data}_{\text{int8}} - \text{offset}')$$

Where,

$\text{data}_{\text{int8}} = \text{data}_{\text{uint8}} - 128 \in [-128, 127], \text{offset}' = -(\text{offset} + 128) \in [-128, 127]$.

Through the foregoing conversion, the data may also be converted into the int8 format. After \textit{scale} and the converted \textit{offset}' are determined, the int8 data converted from the original floating-point data is as follows:

$$\text{data}_{\text{int8}} = \text{clip} \left\{ \text{round} \left( \frac{\text{data}_{\text{float}}}{\text{scale}} \right) + \text{offset}' \right\}$$
10 Operator and Model Fast Query Instructions

10.1 Introduction

Overview
This tool allows you to fast query models and operators supported by CANN of the current version. Supported models are available from ModelZoo of the Ascend Community. Supported operators are available from parsing the OPP in CANN.

NOTE
This tool only queries operators that run on Ascend 310 AI Processor and Ascend 910 AI Processor.

System Requirement
Ubuntu 18.04, CentOS 7.6, or EulerOS 2.8 is supported.

Environment Setup
Set up the development environment by referring to CANN Software Installation Guide.
## 10.2 Instructions

### Command-line Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Example</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-t</td>
<td>Query type:</td>
<td>- op</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>• <strong>op</strong>: operator query</td>
<td>• model</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>model</strong>: model query</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--type</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--opp_path</td>
<td>Installation path of the OPP in CANN Toolkit</td>
<td>/home/xxx/Ascend/</td>
<td>Yes for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ascend-toolkit/latest/</td>
<td>operator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>opp</td>
<td>query</td>
</tr>
<tr>
<td>-o</td>
<td>Path for storing conversion results (in JSON format)</td>
<td>/home/xxx/op.json;</td>
<td>Yes</td>
</tr>
<tr>
<td>--output</td>
<td></td>
<td>/home/xxx/model.json</td>
<td></td>
</tr>
<tr>
<td>-h</td>
<td>Help information</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>--help</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Step 1**  Go to the directory of the script conversion tool.

```
cd {Ascend-CANN-Toolkit installation path}/ascend-toolkit/{version}/{arch}-linux/toolkit/tools/ms_fast_query
```

**Step 2**  Query models.

```
python3.7.5 ms_fast_query.py -t model -o output path
```

**Step 3**  Query operators. In this case, use the absolute path of `ms_fast_query.py`.

```
python3.7.5
{Ascend-CANN-Toolkit installation path}/ascend-toolkit/{version}/{arch}-linux/toolkit/tools/ms_fast_query
ms_fast_query.py -t op --opp_path OPP path -o output path
```

**Step 4**  Complete query.

Check the value of the **result** field in the output file. If the value is **success**, your query is successful.

----End

## 10.3 Query Result

From the output path, you can obtain a JSON list of supported models or operators.
11 Profiling Instructions

11.1 Introduction

11.1.1 Overview

Ascend AI Processor is a high-performance integrated circuit dedicated for AI applications. It consists of Ctrl CPU, AI CPU, and AI Core. Ctrl CPU supports the running of the operating system (OS), while AI CPU and AI Core implement high-performance AI computing. An end-to-end Profiling system can examine the performance of Ascend 910 AI Processor in terms of processor verification, operator development, training, and inference. This system provides an economical solution for achieving optimal performance by accurate location of bottlenecks in software and hardware, efficient analysis, and specific optimization.

The following two profiling modes are supported:

- **Job profiling**
  - Training traces of the AI software stack are collected for analyzing the iterative performance of a training job, including the AI CPU graph compute traces on a single device and the runtime and collective communication traces.
  
  The following metrics can be obtained: iteration elapsed time \( t_{(N+1)6} - t_{N6} \), data augmentation hangover time \( t_{(N+1)1} - t_{N6} \), FPBP elapsed time...
(t_{N2} - t_{N1}), and gradient aggregation and update hangover time (t_{N6} - t_{N2}). Figure 11-1 shows the relationship between these time metrics.

**Figure 11-1** Profiling a training job between iterations

![Diagram showing the relationship between time metrics](image)

**NOTE**

- Data augmentation hangover time: After the last iteration is complete and before the FP of the current iteration starts, data augmentation hangs over for a period of time.
- Gradient aggregation and update hangover time: After the last iteration is complete and before the BP of the current iteration starts, gradient aggregation and update hangs over for a period of time.
  - HWTS and AI Core task traces are collected for analyzing the performance of a training job, including the start time and total time consumption related to the tasks and AI Core.

**System profiling**

Training job–irrelevant profiling is performed on the hardware such as the Ctrl CPU, AI CPU, TS CPU, DVPP, and HBM/DDR for analyzing the system performance such as the CPU usage, memory bandwidth, and PCIe bandwidth.

(Table 11-1 describes the target modules of job profiling and system profiling.

**Table 11-1** Target modules of job profiling and system profiling

<table>
<thead>
<tr>
<th>Category</th>
<th>Node</th>
<th>Module</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job profiling</td>
<td>AI host</td>
<td>Framework</td>
<td>Graph variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HCCL</td>
<td>Collective communication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Runtime</td>
<td>Task traces</td>
</tr>
<tr>
<td></td>
<td>Device</td>
<td>Data preprocessing</td>
<td>Data augmentation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AI Core (task-based)</td>
<td>PMU events</td>
</tr>
<tr>
<td></td>
<td>Task Scheduler track</td>
<td>Task Scheduler timeline</td>
<td></td>
</tr>
</tbody>
</table>
### 11.1.2 Solution Description

Training profiling on the Ascend AI Processor depends on the following CANN components.

<table>
<thead>
<tr>
<th>Kit</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>For development:</td>
<td>Ascend-FwkACLlib</td>
<td>The FwkACLlib component provides the Graph Profiling API to collect job profile data at training time.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Node</th>
<th>Module</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training traces</td>
<td></td>
<td></td>
<td>Iteration traces</td>
</tr>
<tr>
<td>HWTS Log (task-based)</td>
<td></td>
<td></td>
<td>Task elapsed time</td>
</tr>
<tr>
<td>System profiling</td>
<td>Device</td>
<td>Ctrl CPU</td>
<td>PMU events</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AI CPU</td>
<td>PMU events</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TS CPU</td>
<td>PMU events</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HCCS</td>
<td>High-performance inter-chip communication bandwidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LLC</td>
<td>CPU L3 cache</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDR</td>
<td>SDRAM read/write bandwidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Memory</td>
<td>Memory utilization of the system and processes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPU Usage</td>
<td>CPU utilization of the system and processes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HBM</td>
<td>High-bandwidth memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NIC</td>
<td>Network interface card (NIC), error rate, and loss rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RoCE</td>
<td>Ethernet RDMA rate, error rate, and loss rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AI Core (sample-based)</td>
<td>PMU events about AI Core instructions retired, CPU cycles, and more</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PCIe</td>
<td>PCIe read/write bandwidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DVPP</td>
<td>DVPP metrics</td>
</tr>
</tbody>
</table>
### Profiling Instructions

<table>
<thead>
<tr>
<th>Kit</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
</table>
| Ascend-CANN-Toolkit  | Ascend-Toolkit | The Toolkit component contains the profile data collection tool msprof and the profile data parsing tool msprof.py:  
  - msprof: collects system profile data using the command line.  
  - msprof.py: parses profile data collected from job profiling and system profiling using a Python script. |

For training:  
Ascend-CANN-NNAE  
Ascend-FwkACLlib  
The FwkACLlib component provides the Graph Profiling API to collect job profile data at training time.

Choose an installation method that suits your needs:

- Install Ascend-CANN-Toolkit on an Ascend training device, which serves as both the development environment and operating environment.  
  In this environment, both job profiling and system profiling are supported, and the collected profile data can be parsed immediately. This installation method gives you full access to available training features, including operator development, model training, and performance tuning. This installation method is recommended when you need to conduct development activities from code writing to code building and running, as well as code debugging.

- Install Ascend-CANN-NNAE on an Ascend training device, which serves as the operating environment only.  
  In this environment, only job profiling is supported, and system profiling is not supported. This installation method allows you to perform model training. To parse the profile data collected during the training process, transfer it to the environment where Ascend-CANN-Toolkit is installed. If model performance analysis is needed, this installation method is not your best choice.

- Install Ascend-CANN-Toolkit on a non-Ascend AI device, which serves as the development environment only.  
  In this environment, you can only use the msprof.py tool to parse collected profile data.

**NOTE**

- Profile data collection and parsing in a container is supported. The collection mode depends on the CANN software package installed in the container. For details, see Table 11-2.

- In the container scenario, you need to specify the path for flushing profile data to the disk and the path mapped from the host to the container. In the ModelArts scenario, you need to mount the OBS path to the server where the development environment is located so that you can extract profile data for parsing.
11.2 Restrictions

Note the following restrictions when using the Profiling tool:

1. It is not allowed to initiate two profiling tasks on the same device.
2. Paths related to profiling cannot contain special characters.
3. To flush the profile data of a single profiling task to disk with the profiling switches of all metrics turned on, pay attention to the disk read/write speed. If your training job runs on a single device, the disk read/write speed must be greater than or equal to 60 MB/s. If your training job runs on multiple devices, the disk read/write speed must be greater than or equal to that of a single device multiplied by the number of devices.

11.3 Profiling Workflow

Figure 11-2 shows the overall profiling workflow. Set up the environment and deliver a training job with profiling enabled. Profile data is collected during the training process. After the training is complete, you can parse the collected profile data.

Figure 11-2 Profiling workflow
### Table 11-3 Profiling workflow

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set up environment</td>
<td>Set up the environment for collecting and parsing profile data. For details, see 11.4 Environment Setup.</td>
</tr>
<tr>
<td>Job profiling</td>
<td>Collect profile data of a training job. For details, see 11.5.1 Job Profiling.</td>
</tr>
<tr>
<td>System profiling</td>
<td>Collect profile data of hardware data. For details, see 11.5.2 System Profiling.</td>
</tr>
<tr>
<td>Parse profile data</td>
<td>Parse the collected profile data and optimize the performance accordingly. For details, see 11.6 Parsing Profile Data.</td>
</tr>
</tbody>
</table>

**CAUTION**
Profile data cannot be flushed to the disk if the disk is full during profiling. Therefore, it is necessary to have enough space on the disk. In addition, the profile data needs to be aged by the user to prevent the disk space from being used up.

## 11.4 Environment Setup

Refer to 11.1.2 Solution Description for environment setup considerations.

- Method 1: Install Ascend-CANN-Toolkit for inference on an Ascend AI device.
  a. Set up the development environment by referring to the CANN Software Installation Guide. Ensure that Ascend-CANN-Toolkit has been installed.
- Method 2: Install Ascend-CANN-NNAE for training on an Ascend AI device.
  a. Set up the operating environment for collecting job profile data by referring to the CANN Software Installation Guide.
  b. Set up the development environment and install Ascend-CANN-Toolkit for parsing data by referring to the CANN Software Installation Guide.
  a. Install Ascend-CANN-Toolkit by referring to "Installing the Development Kit" in the CANN Software Installation Guide.

## 11.5 Collecting Profile Data

### 11.5.1 Job Profiling

The basic workflows for collecting profile data of a training job are similar no matter the training job is delivered from a physical machine, a container, or ModelArts. However, in the container and ModelArts scenarios, ensure that the collected profile data can be extracted and transferred to the development
environment for parsing. Currently, the following three job profiling methods are available:

1. Start job profiling by editing the training script in either of the following ways (applicable to the TensorFlow framework only):
   - In **Estimator** mode, use `profiling_config` under `NPURunConfig` to enable profiling. The sample code is as follows.
     ```python
     from npu_bridge.estimator.npu.npu_config import NPURunConfig
     from npu_bridge.estimator.npu.npu_config import ProfilingConfig
     profiling_options = {
         "output": "/tmp/profiling",
         "training_trace": "on",
         "fp_point": "resnet_model/conv2d/Conv2Dresnet_model/batch_normalization/FusedBatchNormV3_Reduce",
         "bp_point": "gradients/AddN_70"
     }
     profiling_config = ProfilingConfig(enable_profiling=True, profiling_options=profiling_options)
     session_config=tf.ConfigProto()
     config = NPURunConfig(profiling_config=profiling_config, session_config=session_config)
     ```
   - In `sess.run` mode, use the session configuration options `profiling_mode` and `profiling_options` to enable profiling. The sample code is as follows.
     ```python
     custom_op = config.graph_options.rewrite_options.custom_optimizers.add()
     custom_op.name = "NpuOptimizer"
     custom_op.parameter_map["use_off_line"].b = True
     custom_op.parameter_map["profiling_mode"].b = True
     custom_op.parameter_map["profiling_options"].s = tf.comapt.as_bytes(
         
         
         {
             "output": "/tmp/profiling",
             "training_trace": "on",
             "task_trace": "on",
             "aicpu": "on",
             "fp_point": "resnet_model/conv2d/Conv2Dresnet_model/batch_normalization/FusedBatchNormV3_Reduce",
             "bp_point": "gradients/AddN_70",
             "aic_metrics": "PipeUtilization"
         }
     )
     config.graph_options.rewrite_options.remapping = RewriterConfig.OFF # Disable remapping.
     with tf.Session(config=config) as sess:
         sess.run()
     ```

   For details about `profiling_options`, see **11.8.1 Profiling Options**.

2. Start job profiling by using APIs.
   This method applies to non-migrated, newly developed training networks.
   - Pass the following `option` arguments to the **GEInitialize** API call:
     ```
     ge.exec.profilingMode
     ge.exec.profilingOptions
     ```
   - Then, start profiling by using the following API calls: `aclgrphProfInit`, `aclgrphProfFinalize`, `aclgrphProfCreateConfig`, `aclgrphProfDestroyConfig`, and `aclgrphProfStop`.

   To collect iteration traces, also pass the `ge.exec.profilingOptions` argument to the **GEInitialize** API call or set the `PROFILING_OPTIONS` environment variable. The required fields include `training_trace`, `bp_point`, and `fp_point`.

   For details about the APIs, see **Ascend Graph Development Guide**.

3. Start job profiling by setting environment variables. The following is an example.
   ```
   export PROFILING_MODE=true
   export PROFILING_OPTIONS="{"output": "/tmp/profiling",
                                "training_trace": "on",
                                "task_trace": "on",
                                "aicpu": "on",
                                "fp_point": "resnet_model/conv2d/Conv2dresnet_model/batch_normalization/FusedBatchNormV3_Reduce",
                                "bp_point": "gradients/AddN_70",
                                "aic_metrics": "PipeUtilization""
   ```

   For details about available options, see **11.8.1 Profiling Options**.

Transfer the collected profile data to the development environment for parsing.
For profiling of MindSpore networks, visit the MindSpore website.
To obtain accurate iteration data in the training scenario, you are advised to set `training_trace` in the Profiling options to `on`.

### 11.5.2 System Profiling

Profile data in system profiling is collected by using the `msprof` executable file. The following table describes the functionality and path of the tool.

<table>
<thead>
<tr>
<th>Name</th>
<th>Functionality</th>
<th>Path</th>
</tr>
</thead>
</table>

Before profile data collection, set up the environment by referring to method 2 described in 11.4 Environment Setup.

**Procedure**

**Step 1** Log in to the operating environment as the Ascend-CANN-Toolkit running user. The following assumes that the user is `HwHiAiUser`.

**Step 2** Go to the directory where the `msprof` file is located, for example, `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/bin`.

**Step 3** Collect profile data over software components and hardware modules.

The command-line syntax is described below. Table 11-5 describes the available command-line options.

```
./msprof --output=<dir> --sys-devices=<ID> --sys-period=<period> --sys-hardware-mem=<on/off> --sys-cpu-profiling=<on/off> .......
```

Example:

```
./msprof --output=/home/HwHiAiUser --sys-devices=1 --sys-period=100 --sys-hardware-mem=on
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>--output</td>
<td>Path for storing the collected profile data.</td>
<td>Yes</td>
</tr>
<tr>
<td>--ai-core</td>
<td>AI Core profiling switch, either on or off (default).</td>
<td>No</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Required</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>--aic-mode</td>
<td>AI Core profiling mode, either task-based (default) or sample-based. In task-based mode, profile data is collected task by task; in sample-based mode, profile data is collected at a fixed interval. This option must be used in conjunction with --ai-core set to <strong>on</strong>.</td>
<td>No</td>
</tr>
<tr>
<td>--aic-freq</td>
<td>Sampling frequency (Hz) in sample-based profiling. Defaults to <strong>10</strong>. Must be in the range [10, 1000]. This option must be used in conjunction with --ai-core set to <strong>on</strong>.</td>
<td>No</td>
</tr>
<tr>
<td>--aic-metrics</td>
<td>AI Core metric to profile, selected from ArithmeticUtilization, PipeUtilization (default), Memory, MemoryL0, and ResourceConflictRatio. This option must be used in conjunction with --ai-core set to <strong>on</strong>.</td>
<td>No</td>
</tr>
<tr>
<td>--sys-period</td>
<td>System sampling period (s), at least <strong>0</strong>.</td>
<td>Yes</td>
</tr>
<tr>
<td>--sys-devices</td>
<td>Device ID, required in system profiling. Value: <strong>all</strong> or device IDs separated by commas.</td>
<td>Yes</td>
</tr>
<tr>
<td>--sys-hardware-mem</td>
<td>DDR, HBM, and LLC read/write bandwidth profiling switch, either <strong>on</strong> or <strong>off</strong> (default).</td>
<td>No</td>
</tr>
<tr>
<td>--sys-hardware-mem-freq</td>
<td>DDR, HBM, and LLC sampling frequency (Hz). Defaults to <strong>50</strong>. Must be in the range [1, 1000]. This option must be used in conjunction with --sys-hardware-mem set to <strong>on</strong>.</td>
<td>No</td>
</tr>
<tr>
<td>--llc-profiling</td>
<td>LLC events to profile. <strong>read</strong>: read events, that is, the L2 cache read rate. <strong>write</strong>: write events, that is, the L2 cache write rate. Defaults to <strong>read</strong>. This option must be used in conjunction with --sys-hardware-mem set to <strong>on</strong>.</td>
<td>No</td>
</tr>
<tr>
<td>--sys-cpu-profiling</td>
<td>Profiling switch of CPU (AI CPU, Ctrl CPU, and TS CPU) hotspot functions and PMU events, either <strong>on</strong> or <strong>off</strong> (default).</td>
<td>No</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Required</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>--sys-cpu-freq</td>
<td>Sampling frequency (Hz) for profiling CPU (AI CPU, Ctrl CPU, and TS CPU) hotspot functions and PMU events. Defaults to 50. Must be in the range [1, 50]. This option must be used in conjunction with --sys-cpu-profiling set to on.</td>
<td>No</td>
</tr>
<tr>
<td>--sys-profiling</td>
<td>Profiling switch for system CPU usage and system memory, either on or off (default).</td>
<td>No</td>
</tr>
<tr>
<td>--sys-sampling-freq</td>
<td>Sampling frequency (Hz) for profiling system CPU usage and system memory. Defaults to 10. Must be in the range [1, 10]. This option must be used in conjunction with --sys-profiling set to on.</td>
<td>No</td>
</tr>
<tr>
<td>--sys-pid-profiling</td>
<td>Profiling switch for process CPU usage and process memory, either on or off (default).</td>
<td>No</td>
</tr>
<tr>
<td>--sys-pid-sampling-freq</td>
<td>Sampling frequency (Hz) for profiling process CPU usage and process memory. Defaults to 10. Must be in the range [1, 10]. This option must be used in conjunction with --sys-pid-profiling set to on.</td>
<td>No</td>
</tr>
<tr>
<td>--sys-io-profiling</td>
<td>NIC and RoCE profiling switch, either on or off (default).</td>
<td>No</td>
</tr>
<tr>
<td>--sys-io-sampling-freq</td>
<td>NIC and RoCE sampling frequency (Hz). Defaults to 100. Must be in the range [1, 100]. This option must be used in conjunction with --sys-io-profiling set to on.</td>
<td>No</td>
</tr>
<tr>
<td>--sys-interconnection-profiling</td>
<td>PCIe and HCCS profiling switch, either on or off (default).</td>
<td>No</td>
</tr>
<tr>
<td>--sys-interconnection-freq</td>
<td>PCIe and HCCS sampling frequency (Hz). Defaults to 50. Must be in the range [1, 50]. This option must be used in conjunction with --sys-interconnection-profiling set to on.</td>
<td>No</td>
</tr>
<tr>
<td>--dvpp-profiling</td>
<td>DVPP profiling switch, either on or off (default).</td>
<td>No</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Required</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>--dvpp-freq</td>
<td>DVPP sampling frequency (Hz). Defaults to 50. Must be in the range [1, 100]. This option must be used in conjunction with --dvpp-profiling set to on.</td>
<td>No</td>
</tr>
<tr>
<td>--host-sys</td>
<td>Profiling option (or options separated by commas) on the host, selected from cpu, mem, disk, network, and osrt. This option must be used in conjunction with host-sys-pid. The arguments are as follows:</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>● cpu: process CPU utilization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● mem: process memory utilization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● disk: process disk I/O utilization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● osrt: process syscalls and pthreadcall</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● network: system network I/O utilization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: --host-sys=cpu,mem,disk,network</td>
<td></td>
</tr>
<tr>
<td>--host-sys-pid</td>
<td>PID of the application to profile.</td>
<td>No</td>
</tr>
</tbody>
</table>

**NOTE**

- To collect host-side disk profile data, install the third-party open-source tool iotop. To collect osrt profile data, install the third-party open-source tools perf and ltrace. For details, see [11.8.3 Installing perf, iotop, and ltrace](#).
- Using ltrace to collect the osrt profile data may cause high CPU usage. In addition, using this tool is related to the application’s pthread locking and unlocking, which may affect the process running speed.
- iotop depends on the native Python environment. Therefore, you should collect the host-side disk profile data in the native Python environment.
- In system profiling, --sys-period, --sys-devices, and --output are all required.

**Step 4** After the preceding command is executed, find a JOBXXX directory containing the collected profile data generated in the path specified by the --output option.

**NOTE**

If both device and host data is profiled, two JOBXXX directories are generated in the path specified by --output. Choose the profiling result as prompted.

---End

# 11.6 Parsing Profile Data
**11.6.1 Introduction to msprof.py**

`msprof.py` is a Python-based command-line profiling tool. The following table describes the functionality and path of the tool.

<table>
<thead>
<tr>
<th>Name</th>
<th>Functionality</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>msprof.py</td>
<td>Parses profile data collected in system profiling or job profiling.</td>
<td><code>/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof</code></td>
</tr>
</tbody>
</table>

**NOTE**
- The following assumes that the Profiling installation directory is `/home/HwHiAiUser/Ascend`.
- Unless otherwise specified, the Profiling tool is run by the common user created during tool installation, for example, `HwHiAiUser`.

**11.6.2 Job Profiling**

**11.6.2.1 Parsing Profile Data**

Before parsing profile data in any directory, finish profile data collection by referring to **11.5.1 Job Profiling**.

**Step 1** Log in to the environment as the Ascend-CANN-Toolkit running user. The following assumes that the user is `HwHiAiUser`.

**Step 2** Go to the directory of the `msprof.py` script, for example, `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof`.

**NOTE**
Quick tip: Create an alias for the `msprof.py` script with the command `alias msprof="python3.7.5 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof/msprof.py"` as the `HwHiAiUser` user. Then, you can start profiling with the shortcut `msprof` in any directory.

**Step 3** Parse profile data in any directory in either of the following ways:
- Monitor and parse profile data in any directory. **Table 11-6** describes the available command-line options.
  
  Example:
  
  ```python3.7.5 msprof.py monitor [-h] -dir <dir>```
  
  ```python3.7.5 msprof.py monitor -dir /home/HwHiAiUser/profiler_data```
- Parse profile data in any directory. **Table 11-6** describes the available command-line options.
  
  Example:
  
  ```python3.7.5 msprof.py import [-h] -dir <dir>```
  
  ```python3.7.5 msprof.py import -dir /home/HwHiAiUser/profiler_data```
### Table 11-6 Command-line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Help information, which allows you to find help.</td>
<td>No</td>
</tr>
<tr>
<td>-dir, --collection-dir</td>
<td>Directory of the collected profile data, which contains the JOBXXX directory. You can also pass the JOBXXX directory directly.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**NOTE**

1. When you run the `import` command to parse profile data, a new .db file is generated even if the profile data directory contains an existing .db file. However, before you run the `monitor` command to re-parse the profile data in any directory, delete the existing .db file in the profile data directory.

2. The `monitor` command parses data using multiple processes and is therefore especially useful when the profile data is large.

3. The `monitor` processes do not exit when the script is executed, which means if new profile data is added to the `-dir` folder, the data will be automatically parsed. However, the `import` command parses only existing profile data.

#### Step 4

Find the parsing result file (.db) in the sqlite sub-directory generated under the JOBXXX directory.

#### 11.6.2.2 Querying Training Job Details

Start the query as follows:

**Step 1** Log in to the development environment as the Ascend-CANN-Toolkit running user. The following assumes that the user is HwHiAiUser.

**Step 2** Go to the directory of the `msprof.py` script, for example, `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof`.

**NOTE**

Quick tip: Create an alias for the `msprof.py` script with the command `alias msprof='python3.7.5 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof/msprof.py'` as the HwHiAiUser user. Then, you can start profiling with the shortcut `msprof` in any directory.

**Step 3** Query the job details.

The command-line syntax is described below. **Table 11-7** describes the available command-line options.

```
python3.7.5 msprof.py query [-h] -dir <dir>
```

Example:
python3.7.5 msprof.py query -dir /home/HwHiAiUser/profiler_data

Table 11-7 Command-line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Help information, which allows you to find help.</td>
<td>No</td>
</tr>
<tr>
<td>-dir, --collection-dir</td>
<td>Directory of the collected profile data, which contains the JOBXXX directory. You can also pass the JOBXXX directory directly.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Step 4 After the preceding command is executed, find the query result displayed, as shown in the figure.

The job info, device ID, profile data directory, and collection time are displayed.

11.6.2.3 Obtaining Timeline Reports

11.6.2.3.1 Exporting Timeline Reports

Before exporting timeline reports, finish profile data parsing by referring to 11.6.2.1 Parsing Profile Data. Start the export as follows:

Step 1 Log in to the development environment as the Ascend-CANN-Toolkit running user. The following assumes that the user is HwHiAiUser.

Step 2 Go to the directory of the msprof.py script, for example, /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof.

**NOTE**

Quick tip: Create an alias for the msprof.py script with the command `alias msprof='python3.7.5 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof/msprof.py'` as the HwHiAiUser user. Then, you can start profiling with the shortcut `msprof` in any directory.

Step 3 Export the timeline reports.

The command-line syntax is described below. Table 11-8 describes the available command-line options.

```
python3.7.5 msprof.py export timeline [-h] [-dir <dir>] [--iteration-id <iteration_id>]
```

Example:
python3.7.5 msprof.py export timeline -dir /home/HwHiAiUser/profiler_data --iteration-id 4

Table 11-8 Command-line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Help information, which allows you to find help.</td>
<td>No</td>
</tr>
<tr>
<td>-dir, --collection-dir</td>
<td>Directory of the collected profile data, which contains the JOBxxx directory. You can also pass the JOBxxx directory directly.</td>
<td>Yes</td>
</tr>
<tr>
<td>--iteration-id</td>
<td>Iteration ID. The default value is 1.</td>
<td>No</td>
</tr>
</tbody>
</table>

Step 4 Find a timeline folder containing timeline reports in JSON format generated under the JOBxxx directory of collection-dir. See Table 11-9 for more details.

Table 11-9 Timeline reports

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_time_{device_id}_{iter_id}.json</td>
<td>AI Core, AI CPU, and AllReduce parallelism degree. The JSON file is generated based on profile data files prefixed with at least one of the following fields: aicore, DATA_PREPROCESS.dev.AICPU, and training_trace. For details, see 11.6.2.3.2 Degree of Parallelism of AI Core, AI CPU, and AllReduce.</td>
</tr>
<tr>
<td>training_trace_{device_id}_{iter_id}.json</td>
<td>Iteration traces (iteration elapsed time). The JSON file is generated based on profile data files prefixed with the training_trace field. For details, see 11.6.2.3.3 Iteration Traces.</td>
</tr>
<tr>
<td>ge_{device_id}_{iter_id}.json</td>
<td>Time spent by GE API calls. The file is generated based on profile data files prefixed with the Framework. field. For details, see 11.6.2.3.4 Time Spent by GE API Calls.</td>
</tr>
</tbody>
</table>
**NOTE**

- The timeline reports are generated based on the collected profile data. If the necessary profile data file is absent, the corresponding timeline report will not be available.
- You can run the `export` command to directly export timeline reports from the profile data parsing result. Even the profile data has not been parsed, the `export` command can parse the profile data and export timeline reports.
- To view the generated .json files, access `chrome://tracing` in the Chrome browser and drop the files in blank space.

-----End

### 11.6.2.3.2 Degree of Parallelism of AI Core, AI CPU, and AllReduce

Obtain the `task_time_{device_id}_{iter_id}.json` file by referring to **11.6.2.3.1 Exporting Timeline Reports**. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows (AI CPU is used as an example).

See [Table 11-10](#) for more details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>API name of a component</td>
</tr>
<tr>
<td>Start</td>
<td>Start point on the timeline, which is automatically aligned with that in chrome trace</td>
</tr>
<tr>
<td>Wall Duration</td>
<td>Time spent on the calls to an API (ms)</td>
</tr>
<tr>
<td><strong>AI CPU</strong></td>
<td></td>
</tr>
<tr>
<td><code>compute_time(us)</code></td>
<td>Compute time (μs)</td>
</tr>
<tr>
<td><code>memcpy_time(us)</code></td>
<td>Time taken to copy memory (μs)</td>
</tr>
<tr>
<td><code>task_time(us)</code></td>
<td>Time taken to execute a task (μs)</td>
</tr>
<tr>
<td><code>dispatch_time(us)</code></td>
<td>Dispatch time (μs)</td>
</tr>
<tr>
<td><code>total_time(us)</code></td>
<td>Execution time of a task (μs)</td>
</tr>
<tr>
<td><strong>ALL REDUCE</strong></td>
<td></td>
</tr>
<tr>
<td><code>Reduce Duration(us)</code></td>
<td>Collective communication time (μs)</td>
</tr>
<tr>
<td><strong>TASK SCHEDULER</strong></td>
<td></td>
</tr>
<tr>
<td><code>task_type</code></td>
<td>Task type</td>
</tr>
<tr>
<td><code>stream_id</code></td>
<td>Stream ID</td>
</tr>
</tbody>
</table>
### 11.6.2.3.3 Iteration Traces

Obtain the `training_trace_{device_id}_{iter_id}.json` file by referring to 11.6.2.3.1 Exporting Timeline Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

- **Title**: API name of a component. In this example, the Reduce_1_0 API of Process 1 is selected.
- **Start**: Start point on the timeline, which is automatically aligned with that in chrome trace.
- **Wall Duration**: Time taken by the calls to an API (ms).
- **Iteration End**: End time of the last gradient aggregation of an iteration.
- **Iteration Time**: Iteration elapsed time (= Current Iteration End – Previous Iteration End). The elapsed time of iteration 0 is calculated as Current Iteration End – Current FP Start because the previous Iteration End is absent.
- **FP_BP Time**: FP/BP elapsed time (= BP End – FP Start)

---

**Table 11-11 Field description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>API name of a component. In this example, the Reduce_1_0 API of Process 1 is selected.</td>
</tr>
<tr>
<td>Start</td>
<td>Start point on the timeline, which is automatically aligned with that in chrome trace.</td>
</tr>
<tr>
<td>Wall Duration</td>
<td>Time taken by the calls to an API (ms)</td>
</tr>
<tr>
<td>Iteration End</td>
<td>End time of the last gradient aggregation of an iteration</td>
</tr>
<tr>
<td>Iteration Time</td>
<td>Iteration elapsed time (= Current Iteration End – Previous Iteration End). The elapsed time of iteration 0 is calculated as Current Iteration End – Current FP Start because the previous Iteration End is absent.</td>
</tr>
<tr>
<td>FP_BP Time</td>
<td>FP/BP elapsed time (= BP End – FP Start)</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration ID</td>
<td>Iteration ID</td>
</tr>
<tr>
<td>FP Start</td>
<td>FP start time</td>
</tr>
<tr>
<td>BP End</td>
<td>BP end time</td>
</tr>
<tr>
<td>Grad_refresh Bound</td>
<td>Gradient update hangover time (= <strong>Iteration End</strong> − <strong>BP End</strong>)</td>
</tr>
<tr>
<td>Data_aug Bound</td>
<td>Data augmentation hangover time (= Current <strong>FP Start</strong> − Previous <strong>Iteration End</strong>). The elapsed time of iteration 0 is N/A because the previous <strong>Iteration End</strong> is absent.</td>
</tr>
<tr>
<td>Reduce</td>
<td>Collective communication elapsed time (may involve groups of iterations). <strong>ph:B</strong> indicates the start time, and <strong>ph:E</strong> indicates the end time. If there is only one device, no <strong>Reduce</strong> data is output.</td>
</tr>
</tbody>
</table>

### 11.6.2.3.4 Time Spent by GE API Calls

Obtain the `ge_{device_id}_{iter_id}.json` file by referring to **11.6.2.3.1 Exporting Timeline Reports**. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The following shows a sample report opened with Chrome.

See the following table for more details.

**Table 11-12 Field description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>API name of a component. In this example, the <strong>Output</strong> API of <strong>Thread 4671</strong> is selected.</td>
</tr>
<tr>
<td>Start</td>
<td>Start point on the timeline, which is automatically aligned with that in chrome trace</td>
</tr>
<tr>
<td>Wall Duration</td>
<td>Time taken by the calls to an API (ms)</td>
</tr>
</tbody>
</table>
### 11.6.2.4 Obtaining Summary Reports

#### 11.6.2.4.1 Exporting Summary Reports

Before exporting summary reports, finish profile data parsing by referring to **11.6.2.1 Parsing Profile Data**. Start the export as follows:

**Step 1** Log in to the development environment as the Ascend-CANN-Toolkit running user. The following assumes that the user is `HwHiAiUser`.

**Step 2** Go to the directory of the `msprof.py` script, for example, `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof`.

**NOTE**

Quick tip: Create an alias for the `msprof.py` script with the command `alias msprof='python3.7.5 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof/msprof.py'` as the `HwHiAiUser` user. Then, you can start profiling with the shortcut `msprof` in any directory.

**Step 3** Export the summary reports.

The command-line syntax is described below. **Table 11-13** describes the available command-line options.

```bash
python3.7.5 msprof.py export summary [-h] -dir <dir> [--iteration-id <iteration_id>] [--format <export_format>]
```

Example:

```bash
python3.7.5 msprof.py export summary -dir /home/HwHiAiUser/profiler_data --format csv
```

**Table 11-13 Command-line options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Help information, which allows you to find help.</td>
<td>No</td>
</tr>
<tr>
<td>-dir, --collection-dir</td>
<td>Directory of the collected profile data, which contains the <code>JOBXXX</code> directory. You can also pass the <code>JOBXXX</code> directory directly.</td>
<td>Yes</td>
</tr>
<tr>
<td>--iteration-id</td>
<td>Iteration ID. The default value is 1.</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_id</td>
<td>Model ID.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--format</td>
<td>File format, either csv (default) or json.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong> The following uses summary files in CSV format as examples.</td>
</tr>
</tbody>
</table>

**Step 4** Find a summary folder containing summary reports in CSV format generated under the JOB.XXX directory of collection-dir. See Table 11-14 for more details.

**Table 11-14** Summary reports

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>op_summary_{device_id}_{iter_id}.csv</td>
<td>Time spent by AI Core metrics of a task in an iteration. The CSV file is generated based on profile data files prefixed with the Framework field, the aicore field (optional), and the hwts field. For details, see 11.6.2.4.2 AI Core Summary.</td>
</tr>
<tr>
<td>op_statistic_{device_id}_{iter_id}.csv</td>
<td>AI Core operator statistics table that orders the operator types from top to bottom by execution time. The CSV file is generated based on profile data files prefixed with the Framework field and the hwts field. For details, see 11.6.2.4.3 AI Core Operator Statistics.</td>
</tr>
<tr>
<td>training_trace_{device_id}_{iter_id}.csv</td>
<td>Iteration traces. The CSV file is generated based on profile data files prefixed with the training_trace field. For details, see 11.6.2.4.4 Iteration Traces.</td>
</tr>
<tr>
<td>task_time_{device_id}_{iter_id}.csv</td>
<td>Task Scheduler data. The CSV file is generated based on profile data files prefixed with the hwts field (mandatory). For details, see 11.6.2.4.5 Task Scheduler Summary.</td>
</tr>
<tr>
<td>aicpu_{device_id}_{iter_id}.csv</td>
<td>AI CPU summary. The CSV file is generated based on profile data files prefixed with the DATA_PREPROCESS.dev.AICPU field. For details, see 11.6.2.4.6 AI CPU Summary.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dp_{device_id}_{iter_id}.csv</td>
<td>Data preprocess summary. The CSV file is generated based on profile data files prefixed with the DATA_PREPROCESS.dev.DP. field. For details, see 11.6.2.4.7 Data Augmentation Summary.</td>
</tr>
<tr>
<td>fusion_op_{device_id}_{iter_id}.csv</td>
<td>Operator fusion summary. The CSV file is generated based on profile data files prefixed with the Framework.model_load_info_ field. For details, see 11.6.2.4.8 Operator Fusion Summary.</td>
</tr>
</tbody>
</table>

**NOTE**

- The summary reports are generated based on the collected profile data. If the necessary profile data file is absent, the corresponding summary report will not be available.
- When you open a summary result file in Excel, if a cell or a cell range is displayed in scientific notation, for example, 1.00159E+12, right-click the cell or cell range and select Format Cells… from the shortcut menu. In the displayed Format Cells dialog box, under the Number tab page, select Number and then click OK.
- If a cell is N/A, the data is unavailable.

-----End

11.6.2.4.2 AI Core Summary

Obtain the op_summary_{device_id}_{iter_id}.csv file by referring to 11.6.2.4.1 Exporting Summary Reports. Replace {device_id} with the device ID and {iter_id} with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See the following table for more details.
### Table 11-15 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Name</td>
<td>Model name. It may be left empty if not specified in the collected data.</td>
</tr>
<tr>
<td>Model ID</td>
<td>Model ID</td>
</tr>
<tr>
<td>Task ID</td>
<td>Task ID</td>
</tr>
<tr>
<td>Stream ID</td>
<td>Stream ID</td>
</tr>
<tr>
<td>Infer ID</td>
<td>Inference iteration ID</td>
</tr>
<tr>
<td>Op Name</td>
<td>Operator name</td>
</tr>
<tr>
<td>OP Type</td>
<td>Operator type</td>
</tr>
<tr>
<td>Task Type</td>
<td>Task type</td>
</tr>
<tr>
<td>Task Start Time</td>
<td>Task start time</td>
</tr>
<tr>
<td>Task Duration</td>
<td>Task duration (μs)</td>
</tr>
<tr>
<td>Task Wait Time</td>
<td>Interval between tasks (μs)</td>
</tr>
<tr>
<td>Block Dim</td>
<td>Index of the Core that executes the task</td>
</tr>
<tr>
<td>Input Shapes</td>
<td>Input shapes</td>
</tr>
<tr>
<td>Input Data Types</td>
<td>Input data types</td>
</tr>
<tr>
<td>Input Formats</td>
<td>Input data formats</td>
</tr>
<tr>
<td>Output Shapes</td>
<td>Output shapes</td>
</tr>
<tr>
<td>Output Data Types</td>
<td>Output data types</td>
</tr>
<tr>
<td>Output Formats</td>
<td>Output data formats</td>
</tr>
<tr>
<td>aicore_time</td>
<td>Time taken to execute all task instructions (μs)</td>
</tr>
<tr>
<td>total_cycles</td>
<td>Number of cycles taken to execute all task instructions</td>
</tr>
</tbody>
</table>

For details about the AI Core metrics, see 11.8.2 AI Core Metrics.
If an input is scaler, the corresponding Input Shapes field is empty and formatted as [; ; ; ;]. Each dimension is separated by a semicolon (;).

### 11.6.2.4.3 AI Core Operator Statistics

Obtain the `op_statistic_{device_id}_{iter_id}.csv` file by referring to **11.6.2.4.1 Exporting Summary Reports**. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Name</td>
<td>Model name. It may be left empty if not specified in the collected data.</td>
</tr>
<tr>
<td>OP Type</td>
<td>Operator type</td>
</tr>
<tr>
<td>Core Type</td>
<td>Core type</td>
</tr>
<tr>
<td>Count</td>
<td>Number of operator calls</td>
</tr>
<tr>
<td>Total Time</td>
<td>Time taken by the calls to an operator (μs)</td>
</tr>
<tr>
<td>Avg Time, Min Time, Max Time</td>
<td>Average, minimum, and maximum duration of the operator calls (μs)</td>
</tr>
<tr>
<td>Ratio</td>
<td>Percentage of duration of the operator calls in the model</td>
</tr>
</tbody>
</table>

See the following table for more details.

**Table 11-16 Field description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Name</td>
<td>Model name. It may be left empty if not specified in the collected data.</td>
</tr>
<tr>
<td>OP Type</td>
<td>Operator type</td>
</tr>
<tr>
<td>Core Type</td>
<td>Core type</td>
</tr>
<tr>
<td>Count</td>
<td>Number of operator calls</td>
</tr>
<tr>
<td>Total Time</td>
<td>Time taken by the calls to an operator (μs)</td>
</tr>
<tr>
<td>Avg Time, Min Time, Max Time</td>
<td>Average, minimum, and maximum duration of the operator calls (μs)</td>
</tr>
<tr>
<td>Ratio</td>
<td>Percentage of duration of the operator calls in the model</td>
</tr>
</tbody>
</table>

### 11.6.2.4.4 Iteration Traces

Obtain the `training_trace_{device_id}_{iter_id}.csv` file by referring to **11.6.2.4.1 Exporting Summary Reports**. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.
The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration ID</td>
<td>Iteration ID</td>
</tr>
<tr>
<td>FP Start</td>
<td>FP start time</td>
</tr>
<tr>
<td>BP End</td>
<td>BP end time</td>
</tr>
<tr>
<td>Iteration End</td>
<td>End time of the last gradient aggregation of an iteration</td>
</tr>
<tr>
<td>Iteration Time</td>
<td>Iteration elapsed time (= Current Iteration End – Previous Iteration End). The unit is μs. The elapsed time of iteration 0 is calculated as Current Iteration End – Current FP Start because the previous Iteration End is absent.</td>
</tr>
<tr>
<td>FP to BP Time</td>
<td>FP/BP elapsed time (= BP End – FP Start). The unit is μs.</td>
</tr>
<tr>
<td>Grad_refresh Bound</td>
<td>Gradient update hangover time (= Iteration End – BP End). The unit is μs.</td>
</tr>
<tr>
<td>Data_aug Bound</td>
<td>Data augmentation hangover time (= Current FP Start – Previous Iteration End). The unit is μs. The elapsed time of iteration 0 is N/A because the previous Iteration End is absent.</td>
</tr>
<tr>
<td>Reduce</td>
<td>Total time spent by collective communication. The collective communication duration is divided into two segments according to the default segmentation policy. Reduce Start indicates the start time, and Reduce Duration indicates the duration (μs) from the start to the end. Note that the Reduce columns are not available in a single-device environment.</td>
</tr>
</tbody>
</table>

Iteration tracing is to trace the software status of a training job and the Ascend AI Software Stack, which can be used to analyze the performance of a training job. If the default two-segment gradient segmentation policy is applied, the iteration traces including fp_start, bp_end, Reduce Start, and Reduce Duration(us) of a training job are printed to describe the job execution status in an iteration.
As shown in the figure, to determine the gradient segmentation policy, you need to calculate the difference between \( bp_{end} \) and \( allreduce1_{end} \) as follows:

\[
bp_{end} - allreduce1_{end} = \frac{BP \ End - Reduce \ Start}{100} - \text{Reduce \ Duration}
\]

For details about \( BP \ End \), \( Reduce \ Start \), and \( \text{Reduce \ Duration} \), see Table 11-17. If \( bp_{end} - allreduce1_{end} > 0 \), AR1 is hidden within the BPFP period.

### 11.6.2.4.5 Task Scheduler Summary

Obtain the \texttt{task_time_{device_id}_{iter_id}.csv} file by referring to 11.6.2.4.1 Exporting Summary Reports. Replace \{device_id\} with the device ID and \{iter_id\} with the iteration ID.

The file content is formatted as follows.

See the following table for more details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel_name</td>
<td>Kernel name</td>
</tr>
<tr>
<td>kernel_type</td>
<td>Kernel type</td>
</tr>
<tr>
<td>stream_id</td>
<td>Stream ID</td>
</tr>
<tr>
<td>task_id</td>
<td>Task ID</td>
</tr>
</tbody>
</table>

See the following table for more details.
### 11.6.2.4.6 AI CPU Summary

Obtain the `aicpu_{device_id}_{iter_id}.csv` file by referring to 11.6.2.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Node</th>
<th>Compute_time(ms)</th>
<th>Memcpy_time(ms)</th>
<th>Task_time(ns)</th>
<th>Dispatch_time(ns)</th>
<th>Total_time(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1224964106</td>
<td>Iter</td>
<td>0.258</td>
<td>0.879</td>
<td>1.119852</td>
<td>0.1</td>
<td>1.538</td>
</tr>
</tbody>
</table>

See the following table for more details.

Table 11-19 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>Timestamp of an event</td>
</tr>
<tr>
<td>Node</td>
<td>Node name of a task</td>
</tr>
<tr>
<td>Compute_time</td>
<td>Calculation time taken by a task (ms)</td>
</tr>
<tr>
<td>Memcpy_time</td>
<td>Time taken to copy memory (ms)</td>
</tr>
<tr>
<td>Task_time</td>
<td>Time taken to finish a task (ms)</td>
</tr>
<tr>
<td>Dispatch_time</td>
<td>Time taken to distribute a task (ms)</td>
</tr>
<tr>
<td>Total_time</td>
<td>Total duration (ms)</td>
</tr>
</tbody>
</table>

### 11.6.2.4.7 Data Augmentation Summary

Obtain the `dp_{device_id}_{iter_id}.csv` file by referring to 11.6.2.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Action</th>
<th>Source</th>
<th>Cached Buffer Size</th>
<th>Last queIterator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1224964345</td>
<td></td>
<td></td>
<td>130</td>
<td></td>
</tr>
</tbody>
</table>

The columns in the result file are described as follows.
Table 11-20 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>Timestamp of an event</td>
</tr>
<tr>
<td>Action</td>
<td>Action of an event</td>
</tr>
<tr>
<td>Source</td>
<td>Event source</td>
</tr>
<tr>
<td>Cached Buffer Size</td>
<td>Buffer cache size occupied by an event</td>
</tr>
</tbody>
</table>

11.6.2.4.8 Operator Fusion Summary

Obtain the fusion_op_{device_id}_{iter_id}.csv file by referring to 11.6.2.4.1 Exporting Summary Reports. Replace {device_id} with the device ID and {iter_id} with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Model</th>
<th>Device ID</th>
<th>Stream ID</th>
<th>Original Ops</th>
<th>Model Name</th>
<th>Model ID</th>
<th>Stream ID of the fused operator</th>
<th>Fusion Op</th>
<th>Memory Input</th>
<th>Memory Output</th>
<th>Memory Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>57</td>
<td>1</td>
<td></td>
<td></td>
<td>517_end_traceback</td>
<td>12</td>
<td>404440</td>
<td>404440</td>
<td>404440</td>
</tr>
</tbody>
</table>

See the following table for more details.

Table 11-21 Field description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Name</td>
<td>Model name</td>
</tr>
<tr>
<td>Model ID</td>
<td>Model ID</td>
</tr>
<tr>
<td>Stream ID</td>
<td>Stream ID of the fused operator</td>
</tr>
<tr>
<td>Fusion Op</td>
<td>Fused operator name</td>
</tr>
<tr>
<td>Original Ops</td>
<td>Names of the base operators</td>
</tr>
<tr>
<td>Memory Input</td>
<td>Input memory size, in bytes</td>
</tr>
<tr>
<td>Memory Output</td>
<td>Output memory size, in bytes</td>
</tr>
<tr>
<td>Memory Weight</td>
<td>Weight memory size, in bytes</td>
</tr>
</tbody>
</table>
### 11.6.3 System Profiling

#### 11.6.3.1 Parsing Profile Data in Any Directory

Before parsing profile data in any directory, finish profile data collection by referring to 11.5.2 System Profiling.

**Step 1** Log in to the development environment as the Ascend-CANN-Toolkit running user. The following assumes that the user is HwHiAiUser.

**Step 2** Go to the directory of the `msprof.py` script, for example, `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof`.

**NOTE**

Quick tip: Create an alias for the `msprof.py` script with the command `alias msprof='python3.7.5 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof/msprof.py'` as the HwHiAiUser user. Then, you can start profiling with the shortcut `msprof` in any directory.

**Step 3** Parse profile data in any directory in either of the following ways:

- Monitor and parse profile data in any directory. Table 11-22 describes the available command-line options.

  ```
  python3.7.5 msprof.py monitor [-h] -dir <dir>
  ```

  Example:
  
  ```
  python3.7.5 msprof.py monitor -dir /home/HwHiAiUser/profiler_data
  ```

- Parse profile data in any directory. Table 11-22 describes the available command-line options.

  ```
  python3.7.5 msprof.py import [-h] -dir <dir>
  ```

  Example:
  
  ```
  python3.7.5 msprof.py import -dir /home/HwHiAiUser/profiler_data
  ```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Help information, which allows you to find help.</td>
<td>No</td>
</tr>
</tbody>
</table>

---

### Table 11-22 Command-line options
### Table 11-23

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-dir, --collection-dir</td>
<td>Directory of the collected profile data, which contains the JOBXXX directory. You can also pass the JOBXXX directory directly.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**NOTE**

- When you run the `import` command to parse profile data, a new .db file is generated even if the profile data directory contains an existing .db file. However, before you run the `monitor` command to re-parse the profile data in any directory, delete the existing .db file in the profile data directory.
- The `monitor` command parses data using multiple processes and is therefore especially useful when the profile data is large.
- The `monitor` processes do not exit when the script is executed, which means if new profile data is added to the `-dir` folder, the data will be automatically parsed. However, the `import` command parses only existing profile data.

**Step 4** Find the parsing result file (.db) in the `sqlite` sub-directory generated under the JOBXXX directory.

*****End

### 11.6.3.2 Querying System Profile Data

To query the system profile data, perform the following steps:

**Step 1** Log in to the development environment as the Ascend-CANN-Toolkit running user. The following assumes that the user is HwHiAiUser.

**Step 2** Go to the directory of the `msprof.py` script, for example, `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof`.  

**NOTE**

Quick tip: Create an alias for the `msprof.py` script with the command `alias msprof='python3.7.5 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof/msprof.py'` as the HwHiAiUser user. Then, you can start profiling with the shortcut `msprof` in any directory.

**Step 3** Query the job information.

The command-line syntax is described below. Table 11-23 describes the available command-line options.

```
python3.7.5 msprof.py query [-h] -dir <dir>
```

Example:

```
python3.7.5 msprof.py query -dir /home/HwHiAiUser/profiler_data
```
### Table 11-23 Command-line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h, --help</code></td>
<td>Help information, which allows you to find help.</td>
<td>No</td>
</tr>
<tr>
<td><code>-dir, --collection-dir</code></td>
<td>Directory of the collected profile data, which contains the JOBXXX directory. You can also pass the JOBXXX directory directly.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Step 4
After the preceding command is executed, find the query result displayed, as shown in the figure.

The job info, device ID, profile data directory, and collection time are displayed.

----End

### 11.6.3.3 Obtaining Timeline Reports

#### 11.6.3.3.1 Exporting Timeline Reports

Before exporting timeline reports, finish profile data parsing by referring to 11.6.3.1 Parsing Profile Data in Any Directory. Start the export as follows:

**Step 1**
Log in to the development environment as the Ascend-CANN-Toolkit running user. The following assumes that the user is HwHiAiUser.

**Step 2**
Go to the directory of the msprof.py script, for example, `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof`.

**NOTE**
Quick tip: Create an alias for the msprof.py script with the command `alias msprof='python3.7.5 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof/msprof.py'` as the HwHiAiUser user. Then, you can start profiling with the shortcut `msprof` in any directory.

**Step 3**
Export the timeline reports.

The command-line syntax is described below. **Table 11-24** describes the available command-line options.

```
python3.7.5 msprof.py export timeline [-h] -dir <dir> [--iteration-id <iteration_id>]
```

Example:
```
python3.7.5 msprof.py export timeline -dir /home/HwHiAiUser/profiler_data
```
Table 11-24 Command-line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Help information, which allows you to find help.</td>
<td>No</td>
</tr>
<tr>
<td>-dir, --collection-dir</td>
<td>Directory of the collected profile data, which contains the JOBXXX directory. You can also pass the JOBXXX directory directly.</td>
<td>Yes</td>
</tr>
<tr>
<td>--iteration-id</td>
<td>Iteration ID. The default value is 1.</td>
<td>No</td>
</tr>
</tbody>
</table>

Step 4 Find a timeline folder containing timeline reports in JSON format generated under the JOBXXX directory of collection-dir. See Table 11-25 for more details.

Table 11-25 Timeline reports

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddr_{device_id}_{iter_id}.json</td>
<td>DDR memory read/write speed. The JSON file is generated based on profile data files prefixed with the ddr. field. For details, see 11.6.3.3.2 DDR Memory Read/Write Speed.</td>
</tr>
<tr>
<td>hbm_{device_id}_{iter_id}.json</td>
<td>HBM memory read/write speed. The JSON file is generated based on profile data files prefixed with the hbm. field. For details, see 11.6.3.3.3 HBM Memory Read/Write Speed.</td>
</tr>
<tr>
<td>pcie_{device_id}_{iter_id}.json</td>
<td>PCIe bandwidth. The JSON file is generated based on profile data files prefixed with the pcie. field. For details, see 11.6.3.3.4 PCIe Bandwidth.</td>
</tr>
<tr>
<td>nic_{device_id}_{iter_id}.json</td>
<td>NIC information. The JSON file is generated based on profile data files prefixed with the nic. field. For details, see 11.6.3.3.5 NIC Summary.</td>
</tr>
<tr>
<td>roce_{device_id}_{iter_id}.json</td>
<td>RoCE bandwidth. The JSON file is generated based on profile data files prefixed with the roce. field. For details, see 11.6.3.3.6 RoCE Bandwidth.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>llc_read_write_{device_id}_{iter_id}.json</td>
<td>L3 cache read/write speed. The JSON file is generated based on profile data files prefixed with the llc field. For details, see 11.6.3.3.7 L3 Cache Read/Write Speed.</td>
</tr>
<tr>
<td>hccs_{device_id}_{iter_id}.json</td>
<td>HCCS bandwidth. The JSON file is generated based on profile data files prefixed with the hccs field. For details, see 11.6.3.3.8 HCCS Bandwidth.</td>
</tr>
<tr>
<td>host_cpu_usage.json</td>
<td>CPU utilization on the host. The file is generated based on profile data files prefixed with the host_cpu field. For details, see 11.6.3.3.9 Host CPU Utilization.</td>
</tr>
<tr>
<td>host_disk_usage.json</td>
<td>Disk I/O utilization on the host. The file is generated based on profile data files prefixed with the host_disk field. For details, see 11.6.3.3.10 Host Disk I/O Utilization.</td>
</tr>
<tr>
<td>host_mem_usage.json</td>
<td>Memory utilization on the host. The file is generated based on profile data files prefixed with the host_mem field. For details, see 11.6.3.3.11 Host Memory Utilization.</td>
</tr>
<tr>
<td>host_network_usage.json</td>
<td>Network I/O utilization on the host. The file is generated based on profile data files prefixed with the host_network field. For details, see 11.6.3.3.12 Host Network I/O Utilization.</td>
</tr>
<tr>
<td>os_runtime_api.json</td>
<td>syscall and pthreadcall data on the host. The file is generated based on profile data files prefixed with the host_syscall or host_pthreadcall field. For details, see 11.6.3.3.13 Host syscall and pthreadcall.</td>
</tr>
</tbody>
</table>

**NOTE**

- The timeline reports are generated based on the collected profile data. If the necessary profile data file is absent, the corresponding timeline report will not be available.
- You can run the export command to directly export timeline reports from the profile data parsing result. Even the profile data has not been parsed, the export command can parse the profile data and export timeline reports.
- To view the generated .json files, access chrome://tracing in the Chrome browser and drop the files in blank space.

----End
11.6.3.3.2 DDR Memory Read/Write Speed

Obtain the `ddr_{device_id}_{iter_id}.json` file by referring to 11.6.3.3.1 Exporting Timeline Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The following shows a sample report opened with Chrome.

See the following table for more details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR/Read</td>
<td>DDR memory read operation</td>
</tr>
<tr>
<td>DDR/Write</td>
<td>DDR memory write operation</td>
</tr>
<tr>
<td>read(MB/s), write(MB/s)</td>
<td>HBM memory read/write speed</td>
</tr>
<tr>
<td>Time</td>
<td>A point on the timeline</td>
</tr>
<tr>
<td>Value</td>
<td>Values of <code>flux_write(MB/s)</code> and <code>flux_read(MB/s)</code> at a certain point</td>
</tr>
</tbody>
</table>

11.6.3.3.3 HBM Memory Read/Write Speed

Obtain the `hbm_{device_id}_{iter_id}.json` file by referring to 11.6.3.3.1 Exporting Timeline Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The following shows a sample report opened with Chrome.

See the following table for more details.
### Table 11-27 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBM_&lt;id&gt;/Read</td>
<td>HBM memory read operation</td>
</tr>
<tr>
<td>HBM_&lt;id&gt;/Write</td>
<td>HBM memory write operation</td>
</tr>
<tr>
<td>Write(MB/s) , Read(MB/s)</td>
<td>HBM memory read/write speed</td>
</tr>
<tr>
<td>Time</td>
<td>A point on the timeline</td>
</tr>
<tr>
<td>Value</td>
<td>Values of Write(MB/s) and Read(MB/s) at a certain point</td>
</tr>
</tbody>
</table>

### 11.6.3.3.4 PCIe Bandwidth

Obtain the `pcie_{device_id}_{iter_id}.json` file by referring to 11.6.3.3.1 Exporting Timeline Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The following shows a sample report opened with Chrome.

![Sample PCIe Bandwidth Report](image)

See the following table for more details.

### Table 11-28 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIe_post</td>
<td>PCIe posted data transfer mode</td>
</tr>
<tr>
<td>PCIe_nonpost</td>
<td>PCIe non-posted data transfer mode</td>
</tr>
<tr>
<td>PCIe_cpl</td>
<td>Completion packet (Cpl) that receives write requests</td>
</tr>
<tr>
<td>PCIe_nonpost_latency</td>
<td>PCIe non-posted data transfer delay</td>
</tr>
<tr>
<td>Rx, Tx</td>
<td><strong>Tx</strong> stands for the transmit end’s bandwidth whereas <strong>Rx</strong> stands for the receive end’s bandwidth.</td>
</tr>
<tr>
<td>Time</td>
<td>A point on the timeline</td>
</tr>
</tbody>
</table>
### 11.6.3.3.5 NIC Summary

Obtain the `nic_{device_id}_{iter_id}.json` file by referring to [11.6.3.3.1 Exporting Timeline Reports](#). Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The following shows a sample report opened with Chrome.

![Sample NIC Report](image)

See the following table for more details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Values of Rx and Tx at a certain point</td>
</tr>
</tbody>
</table>

### Table 11-29 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tx/rx_bandwidth</code></td>
<td>Packet receive/transmit bandwidth</td>
</tr>
<tr>
<td><code>rx_packets</code></td>
<td>Packet receive/transmit rate</td>
</tr>
<tr>
<td><code>rx_error_rate</code></td>
<td>Received/Transmitted packet error rate</td>
</tr>
<tr>
<td><code>rx_dropped_rate</code></td>
<td>Received/Transmitted packet loss rate</td>
</tr>
<tr>
<td>Time</td>
<td>A point on the timeline</td>
</tr>
<tr>
<td>Value</td>
<td>Value of Series at a certain point</td>
</tr>
</tbody>
</table>

---

### 11.6.3.3.6 RoCE Bandwidth

Obtain the `roce_{device_id}_{iter_id}.json` file by referring to [11.6.3.3.1 Exporting Timeline Reports](#). Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The following shows a sample report opened with Chrome.
See the following table for more details.

### Table 11-30 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx/rx_bandwidth</td>
<td>Packet receive/transmit bandwidth</td>
</tr>
<tr>
<td>h_efficiency</td>
<td></td>
</tr>
<tr>
<td>tx/rx_packets</td>
<td>Packet receive/transmit rate</td>
</tr>
<tr>
<td>tx/rx_error_rate</td>
<td>Received/Transmitted packet error rate</td>
</tr>
<tr>
<td>tx/rx_dropped_rate</td>
<td>Received/Transmitted packet loss rate</td>
</tr>
<tr>
<td>Time</td>
<td>A point on the timeline</td>
</tr>
<tr>
<td>Value</td>
<td>Value of Series at a certain point</td>
</tr>
</tbody>
</table>

### 11.6.3.3.7 L3 Cache Read/Write Speed

Obtain the `llc_read_write_{device_id}_{iter_id}.json` file by referring to **11.6.3.3.1 Exporting Timeline Reports**. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The following shows a sample report opened with Chrome.

See the following table for more details.
### Table 11-31 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLC &lt;id&gt; Read/Throughput</td>
<td>L3 cache read throughput</td>
</tr>
<tr>
<td>LLC &lt;id&gt; Read/Hit Rate</td>
<td>L3 cache read hit rate</td>
</tr>
<tr>
<td>Time</td>
<td>A point on the timeline</td>
</tr>
<tr>
<td>Value</td>
<td>Value of rate at a certain point</td>
</tr>
</tbody>
</table>

#### 11.6.3.3.8 HCCS Bandwidth

Obtain the `hccs_{device_id}_{iter_id}.json` file by referring to **11.6.3.3.1 Exporting Timeline Reports**. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The following shows a sample report opened with Chrome.

See the following table for more details.

### Table 11-32 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCCS Timeline</td>
<td>HCCS timeline data</td>
</tr>
<tr>
<td>Rx(MB/s), Tx(MB/s)</td>
<td>Receive bandwidth and transmit bandwidth</td>
</tr>
<tr>
<td>Time</td>
<td>A point on the timeline</td>
</tr>
<tr>
<td>Value</td>
<td>Values of Rx(MB/s) and Tx(MB/s) at a certain point</td>
</tr>
</tbody>
</table>

#### 11.6.3.3.9 Host CPU Utilization

Obtain the host-side file `host_cpu_usage.json` by referring to **11.6.3.3.1 Exporting Timeline Reports**.

The following shows a sample report opened with Chrome.
See the following table for more details.

### Table 11-33 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU &lt;id&gt;</td>
<td>CPU ID</td>
</tr>
<tr>
<td>CPU Avg</td>
<td>Average CPU utilization</td>
</tr>
<tr>
<td>usage</td>
<td>Utilization</td>
</tr>
<tr>
<td>Time</td>
<td>A point on the timeline</td>
</tr>
<tr>
<td>Value</td>
<td>Value of usage at a certain point</td>
</tr>
</tbody>
</table>

### 11.6.3.3.10 Host Disk I/O Utilization

Obtain the host-side file **host_disk_usage.json** by referring to **11.6.3.3.1 Exporting Timeline Reports**.

The following shows a sample report opened with Chrome.

See the following table for more details.

### Table 11-34 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk Usage</td>
<td>Disk utilization</td>
</tr>
<tr>
<td>Time</td>
<td>A point on the timeline</td>
</tr>
<tr>
<td>Value</td>
<td>Value of usage at a certain point</td>
</tr>
</tbody>
</table>
11.6.3.3.11 Host Memory Utilization

Obtain the host-side file `host_mem_usage.json` by referring to **11.6.3.3.1 Exporting Timeline Reports**.

The following shows a sample report opened with Chrome.

See the following table for more details.

**Table 11-35 Field description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Usage</td>
<td>Memory utilization</td>
</tr>
<tr>
<td>Time</td>
<td>A point on the timeline</td>
</tr>
<tr>
<td>Value</td>
<td>Value of usage at a certain point</td>
</tr>
</tbody>
</table>

11.6.3.3.12 Host Network I/O Utilization

Obtain the host-side file `host_network_usage.json` by referring to **11.6.3.3.1 Exporting Timeline Reports**.

The following shows a sample report opened with Chrome.

See the following table for more details.

**Table 11-36 Field description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Usage</td>
<td>Network I/O utilization</td>
</tr>
</tbody>
</table>
### 11.6.3.3.13 Host syscall and pthreadcall

Obtain the host-side file `os_runtime_api.json` by referring to **11.6.3.1 Exporting Timeline Reports**.

The following shows a sample report opened with Chrome.

See the following table for more details.

**Table 11-37 Field description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>API name of a component. As shown in the figure, <strong>pthread_mutex_unlock</strong> is selected.</td>
</tr>
<tr>
<td>Start</td>
<td>Start point on the timeline, which is automatically aligned with that in chrome trace.</td>
</tr>
<tr>
<td>Wall Duration</td>
<td>Time taken by the calls to an API (ms)</td>
</tr>
</tbody>
</table>

### 11.6.3.4 Obtaining Summary Reports

#### 11.6.3.4.1 Exporting Summary Reports

Before exporting summary reports, finish profile data parsing by referring to **11.5.2 System Profiling**. Start the export as follows:

**Step 1** Log in to the development environment as the Ascend-CANN-Toolkit running user. The following assumes that the user is **HwHiAiUser**.

**Step 2** Go to the directory of the `msprof.py` script, for example, `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof`.
**NOTE**

To run the `msprof` command in any directory, use the command-line syntax below.

```bash
python3.7.5 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/profiler/profiler_tool/analysis/msprof/msprof.py xxx
```

**Step 3** Export the summary reports.

The command-line syntax is described below. **Table 11-38** describes the available command-line options.

```bash
python3.7.5 msprof.py export summary [-h] -dir <dir> [--iteration-id <iteration_id>] [--format <export_format>]
```

Example:

```bash
python3.7.5 msprof.py export summary -dir /home/HwHiAiUser/profiler_data --format csv
```

**Table 11-38** Command-line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Help information, which allows you to find help.</td>
<td>No</td>
</tr>
<tr>
<td>-dir, --collection-dir</td>
<td>Directory of the collected profile data, which contains the <code>JOBXXX</code> directory. You can also pass the <code>JOBXXX</code> directory directly.</td>
<td>Yes</td>
</tr>
<tr>
<td>--iteration-id</td>
<td>Iteration ID. The default value is 1.</td>
<td>No</td>
</tr>
<tr>
<td>--format</td>
<td>File format, either <code>csv</code> (default) or <code>json</code>. <strong>NOTE</strong> The following uses summary files in CSV format as examples.</td>
<td>No</td>
</tr>
</tbody>
</table>

**Step 4** Find a `summary` folder containing summary reports in JSON format generated under the `JOBXXX` directory of `collection-dir`. See **Table 11-39** for more details.

**Table 11-39** Summary reports

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ddr_{device_id}_{iter_id}.csv</code></td>
<td>DDR memory read/write speed. The CSV file is generated based on profile data files prefixed with the <code>ddr</code> field. For details, see 11.6.3.4.2 DDR Memory Read/Write Speed.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>hbm_{device_id}_{iter_id}.csv</td>
<td>HBM memory read/write speed. The CSV file is generated based on profile data files prefixed with the hbm field. For details, see 11.6.3.4.3 HBM Memory Read/Write Speed.</td>
</tr>
<tr>
<td>pcie_{device_id}_{iter_id}.csv</td>
<td>PCIe bandwidth. The CSV file is generated based on profile data files prefixed with the pcie field. For details, see 11.6.3.4.4 PCIe Bandwidth.</td>
</tr>
<tr>
<td>cpu_usage_{device_id}_{iter_id}.csv</td>
<td>AI CPU utilization and Ctrl CPU utilization. The CSV file is generated based on profile data files prefixed with the SystemCpuUsage field. For details, see 11.6.3.4.5 AI CPU and Ctrl CPU Utilization.</td>
</tr>
<tr>
<td>sys_mem_{device_id}_{iter_id}.csv</td>
<td>Memory utilization of a device. The CSV file is generated based on profile data files prefixed with the Memory field. For details, see 11.6.3.4.6 System Memory Utilization.</td>
</tr>
<tr>
<td>ai_core_utilization_{device_id}_{iter_id}.csv</td>
<td>Core utilization by instructions. The CSV file generated in sample-based profiling is based on profile data files prefixed with the aicore field. For details, see 11.6.3.4.7 AI Core Utilization.</td>
</tr>
<tr>
<td>process_cpu_usage_{device_id}_{iter_id}.csv</td>
<td>Process CPU utilization. The CSV file is generated based on profile data files prefixed with the CpuUsage field. For details, see 11.6.3.4.8 Process CPU Utilization.</td>
</tr>
<tr>
<td>process_mem_{device_id}_{iter_id}.csv</td>
<td>Process memory utilization. The CSV file is generated based on profile data files prefixed with the Memory field. For details, see 11.6.3.4.9 Process Memory Utilization.</td>
</tr>
<tr>
<td>hccs_{device_id}_{iter_id}.csv</td>
<td>HCCS bandwidth. The CSV file is generated based on profile data files prefixed with the hccs field. For details, see 11.6.3.4.10 HCCS Bandwidth.</td>
</tr>
<tr>
<td>nic_{device_id}_{iter_id}.csv</td>
<td>NIC summary. The CSV file is generated based on profile data files prefixed with the nic field. For details, see 11.6.3.4.11 NIC Summary.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dvpp_{device_id}_{iter_id}.csv</td>
<td>DVPP summary. The CSV file is generated based on profile data files prefixed with the dvpp. field. For details, see 11.6.3.4.12 DVPP Summary.</td>
</tr>
<tr>
<td>llc_read_write_{device_id}_{iter_id}.csv</td>
<td>L3 cache read/write speed. The CSV file is generated based on profile data files prefixed with the llc. field. For details, see 11.6.3.4.13 L3 Cache Read/Write Speed.</td>
</tr>
<tr>
<td>roce_{device_id}_{iter_id}.csv</td>
<td>RoCE bandwidth. The CSV file is generated based on profile data files prefixed with the roce. field. For details, see 11.6.3.4.14 RoCE Bandwidth.</td>
</tr>
<tr>
<td>ai_cpu_top_function_{device_id}_{iter_id}.csv</td>
<td>AI CPU top functions. The CSV file is generated based on profile data files prefixed with the ai_ctrl_cpu. field. For details, see 11.6.3.4.15 AI CPU Top Functions.</td>
</tr>
<tr>
<td>ai_cpu_pmu_events_{device_id}_{iter_id}.csv</td>
<td>AI CPU PMU events. The CSV file is generated based on profile data files prefixed with the ai_ctrl_cpu. field. For details, see 11.6.3.4.16 AI CPU PMU Events.</td>
</tr>
<tr>
<td>ctrl_cpu_top_function_{device_id}_{iter_id}.csv</td>
<td>Ctrl CPU top functions. The CSV file is generated based on profile data files prefixed with the ai_ctrl_cpu. field. For details, see 11.6.3.4.17 Ctrl CPU Top Functions.</td>
</tr>
<tr>
<td>ctrl_cpu_pmu_events_{device_id}_{iter_id}.csv</td>
<td>Ctrl CPU PMU events. The CSV file is generated based on profile data files prefixed with the ai_ctrl_cpu. field. For details, see 11.6.3.4.18 Ctrl CPU PMU Events.</td>
</tr>
<tr>
<td>ts_cpu_top_function_{device_id}_{iter_id}.csv</td>
<td>TS CPU top functions. The CSV file is generated based on profile data files prefixed with the tscpu. field. For details, see 11.6.3.4.19 TS CPU Top Functions.</td>
</tr>
<tr>
<td>ts_cpu_pmu_events_{device_id}_{iter_id}.csv</td>
<td>TS CPU PMU events. The CSV file is generated based on profile data files prefixed with the tscpu. field. For details, see 11.6.3.4.20 TS CPU PMU Events.</td>
</tr>
</tbody>
</table>
NOTE

- The summary reports are generated based on the collected profile data. If the necessary profile data file is absent, the corresponding summary report will not be available.
- You can run the `export` command to directly export timeline reports from the profile data parsing result. Even the profile data has not been parsed, the `export` command can parse the profile data and export timeline reports.
- When you open a summary result file in Excel, if a cell or a cell range is displayed in scientific notation, for example, `1.00159E+12`, right-click the cell or cell range and select `Format Cells...` from the shortcut menu. In the displayed `Format Cells` dialog box, under the `Number` tab page, select `Number` and then click `OK`.

--- End

11.6.3.4.2 DDR Memory Read/Write Speed

Obtain the `ddr_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Read(MB/s)</th>
<th>Write(MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>119.996770</td>
<td>32.015443</td>
</tr>
</tbody>
</table>

See Table 11-40 for more details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
<td>Metric</td>
</tr>
<tr>
<td>Read</td>
<td>Read bandwidth (MB/s)</td>
</tr>
<tr>
<td>Write</td>
<td>Write bandwidth (MB/s)</td>
</tr>
</tbody>
</table>

11.6.3.4.3 HBM Memory Read/Write Speed

Obtain the `hbm_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Read(MB/s)</th>
<th>Write(MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.095985</td>
<td>3.00E-05</td>
</tr>
<tr>
<td>0</td>
<td>0.095985</td>
<td>3.00E-05</td>
</tr>
<tr>
<td>1</td>
<td>0.095985</td>
<td>3.00E-05</td>
</tr>
<tr>
<td>2</td>
<td>0.095985</td>
<td>3.00E-05</td>
</tr>
<tr>
<td>3</td>
<td>0.095985</td>
<td>3.00E-05</td>
</tr>
</tbody>
</table>

See Table 11-41 for more details.
### 11.6.3.4.4 PCIe Bandwidth

Obtain the `pcie_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>7s_n_avg (MB/s)</code></td>
<td>0</td>
<td>59.9322</td>
<td>0.58330.12</td>
</tr>
<tr>
<td><code>7s_re_avg (MB/s)</code></td>
<td>0.49.97722</td>
<td>0.5606718</td>
<td></td>
</tr>
<tr>
<td><code>7sctp_avg (MB/s)</code></td>
<td>0</td>
<td></td>
<td>0.39785276</td>
</tr>
<tr>
<td><code>7s_latency_avg (MB/s)</code></td>
<td>247.9021</td>
<td>424.6278</td>
<td>257.0464</td>
</tr>
<tr>
<td><code>8s_n_avg (MB/s)</code></td>
<td>0.001222</td>
<td>156.2047</td>
<td>0.60327614</td>
</tr>
<tr>
<td><code>8s_re_avg (MB/s)</code></td>
<td>0.5986</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>8sctp_avg (MB/s)</code></td>
<td>0.149.0139</td>
<td></td>
<td>0.6139634</td>
</tr>
</tbody>
</table>

See Table 11-42 for more details.

### 11.6.3.4.5 AI CPU and Ctrl CPU Utilization

Obtain the `cpu_usage_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Cpu Type</th>
<th>User(%)</th>
<th>Sys(%)</th>
<th>Idle(%)</th>
<th>IoNt(%)</th>
<th>Irq(%)</th>
<th>Soft(%)</th>
<th>Idle(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>slqcpu</td>
<td>0.000133</td>
<td>0.69075</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>95.99611</td>
<td>0.00289</td>
</tr>
<tr>
<td>ctrlcpu</td>
<td>0.045897</td>
<td>0.477637</td>
<td>0</td>
<td>0.003882</td>
<td>98.42767</td>
<td>0.00289</td>
<td></td>
</tr>
</tbody>
</table>
### Table 11-43 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpu Type</td>
<td>CPU type</td>
</tr>
<tr>
<td>User</td>
<td>Percentage of time taken to execute user-mode processes</td>
</tr>
<tr>
<td>Sys</td>
<td>Percentage of time taken to execute kernel-mode processes</td>
</tr>
<tr>
<td>IoWait</td>
<td>Percentage of I/O wait time</td>
</tr>
<tr>
<td>Iirq</td>
<td>Percentage of the hardware interrupt time</td>
</tr>
<tr>
<td>Soft</td>
<td>Percentage of the software interrupt time</td>
</tr>
<tr>
<td>Idle</td>
<td>Percentage of idle time</td>
</tr>
</tbody>
</table>

#### 11.6.3.4.6 System Memory Utilization

Obtain the `sys_mem_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

```plaintext
Memory Total(KB) Memory Free(KB) Buffers(KB) Cache(KB) Share Memory(KB) Commit Limit(KB) Committed AS(KB) Huge Pages Total(pages) Huge Pages Free(pages)
```

See the following table for more details.

### Table 11-44 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Total</td>
<td>Total system memory (KB)</td>
</tr>
<tr>
<td>Memory Free</td>
<td>Available system memory (KB)</td>
</tr>
<tr>
<td>Buffers</td>
<td>Buffers (KB)</td>
</tr>
<tr>
<td>Cached</td>
<td>Cache (KB)</td>
</tr>
<tr>
<td>Share Memory</td>
<td>Shared memory (KB)</td>
</tr>
<tr>
<td>Commit Limit</td>
<td>Commit limit (KB)</td>
</tr>
<tr>
<td>Committed AS</td>
<td>Committed memory (KB)</td>
</tr>
<tr>
<td>Huge Pages Total</td>
<td>Total huge pages</td>
</tr>
<tr>
<td>Huge Pages Free</td>
<td>Free huge pages</td>
</tr>
</tbody>
</table>
11.6.3.4.7 AI Core Utilization

Obtain the ai_core_utilization_{device_id}_{iter_id}.csv file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace {device_id} with the device ID and {iter_id} with the iteration ID.

The file content is formatted as follows.

```
CPU ID      perf指令集      perf指令集      perf指令集      perf指令集      perf指令集
Core0       0               0               0               0               0
Core1       0               0               0               0               0
Core2       0               0               0               0               0
Core3       0               0               0               0               0
Core4       0               0               0               0               0
Core5       0               0               0               0               0
Core6       0               0               0               0               0
Core7       0               0               0               0               0
Core8       0               0               0               0               0
```

See 11.8.2 AI Core Metrics for more details. The AI Core metric ArithmeticUtilization is taken as an example.

11.6.3.4.8 Process CPU Utilization

Obtain the process_cpu_usage_{device_id}_{iter_id}.csv file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace {device_id} with the device ID and {iter_id} with the iteration ID.

The file content is formatted as follows.

```
Field     PID          Name            CPU(%)      
----------  -----------  -----------   -----------
5291 adda   0.02148    
5321 dmp daemon 0.018775   
5111 monitor  0.005858   
1 luft       0.002324    
4082 kworker/165-events 0.003804  
4410 dvolg_work_l  0.001453  
214 kworker/02-events 0.00121  
4408 dvolg_work_0  0.00154  
5122 ascend_monitor  0.000699  
11 rcu_sched  0.000424  
```

See the following table for more details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>Process ID</td>
</tr>
<tr>
<td>Name</td>
<td>Process name</td>
</tr>
<tr>
<td>CPU(%)</td>
<td>CPU utilization of a process</td>
</tr>
</tbody>
</table>

11.6.3.4.9 Process Memory Utilization

Obtain the process_mem_{device_id}_{iter_id}.csv file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace {device_id} with the device ID and {iter_id} with the iteration ID.

The file content is formatted as follows.
See the following table for more details.

**Table 11-46 Field description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>Process ID</td>
</tr>
<tr>
<td>Name</td>
<td>Process name</td>
</tr>
<tr>
<td>Size</td>
<td>Memory pages used by a process</td>
</tr>
<tr>
<td>Resident</td>
<td>Physical memory pages used by a process</td>
</tr>
<tr>
<td>Shared</td>
<td>Shared memory pages used by a process</td>
</tr>
</tbody>
</table>

### 11.6.3.4.10 HCCS Bandwidth

Obtain the `hccs_{device_id}_{iter_id}.csv` file by referring to **11.6.3.4.1 Exporting Summary Reports**. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Max (MB/s)</th>
<th>Min (MB/s)</th>
<th>Average (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx</td>
<td>8972.964</td>
<td>8.089749</td>
<td>577.5706</td>
</tr>
<tr>
<td>Rx</td>
<td>8972.941</td>
<td>8.089749</td>
<td>577.5704</td>
</tr>
</tbody>
</table>

See the following table for more details.

**Table 11-47 Field description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Mode. <strong>Tx</strong> stands for the transmit direction whereas <strong>Rx</strong> standards for the receive direction.</td>
</tr>
<tr>
<td>Max</td>
<td>Minimum bandwidth (MB/s)</td>
</tr>
<tr>
<td>Min</td>
<td>Maximum bandwidth (MB/s)</td>
</tr>
<tr>
<td>Average</td>
<td>Average bandwidth (MB/s)</td>
</tr>
</tbody>
</table>
NOTE

If your target device has only one Ascend AI Processor, this HCCS report is unavailable.

11.6.3.4.11 NIC Summary

Obtain the `nic_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>Time</td>
</tr>
<tr>
<td>Bandwidth(MB/s)</td>
<td>Bandwidth (MB/s)</td>
</tr>
<tr>
<td>Rx Bandwidth efficiency</td>
<td>Receive bandwidth</td>
</tr>
<tr>
<td>rxPacket/s</td>
<td>Packet receive rate</td>
</tr>
<tr>
<td>rxError rate</td>
<td>Received packet error rate</td>
</tr>
<tr>
<td>rxDropped rate</td>
<td>Received packet loss rate</td>
</tr>
<tr>
<td>Tx Bandwidth efficiency</td>
<td>Transmit bandwidth</td>
</tr>
<tr>
<td>txPacket/s</td>
<td>Packet transmit rate</td>
</tr>
<tr>
<td>txError rate</td>
<td>Transmitted packet error rate</td>
</tr>
<tr>
<td>txDropped rate</td>
<td>Transmitted packet loss rate</td>
</tr>
<tr>
<td>funcId</td>
<td>NIC ID</td>
</tr>
</tbody>
</table>

See the following table for more details.

Table 11-48 Field description

11.6.3.4.12 DVPP Summary

Obtain the `dvpp_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Dvpp Id</th>
<th>Engine Type</th>
<th>Engine ID</th>
<th>All Time(us)</th>
<th>All Frame</th>
<th>All Utilization(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 VDEC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 VDEC</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 VDEC</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 VDEC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 JPRGD</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 JPRGD</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 JPRGD</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 JPRGD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 MRGD</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 MRGD</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 MRGD</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
See the following table for more details.

### Table 11-49 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dvpp Id</td>
<td>DVPP ID.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong></td>
</tr>
<tr>
<td></td>
<td>The ID range is related to the number of devices on the AI Server. Each device is mapped to five IDs, indexed starting at 0.</td>
</tr>
<tr>
<td>Engine Type</td>
<td>Engine type</td>
</tr>
<tr>
<td>Engine ID</td>
<td>Engine ID</td>
</tr>
<tr>
<td>All Time(us)</td>
<td>Time taken by each sampling (μs)</td>
</tr>
<tr>
<td>All Frame</td>
<td>Number of frames processed by the engine each sampling</td>
</tr>
<tr>
<td>All Utilization</td>
<td>Average utilization, that is, the total busy time divided by the total up time</td>
</tr>
</tbody>
</table>

### 11.6.3.4.13 L3 Cache Read/Write Speed

Obtain the `llc_read_write_{device_id}_{iter_id}.csv` file by referring to **11.6.3.4.1 Exporting Summary Reports**. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Task</th>
<th>Hit Rate</th>
<th>Throughput(MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>Average</td>
<td>0.955799</td>
<td>279021888.4</td>
</tr>
<tr>
<td>read</td>
<td>1</td>
<td>0.94942</td>
<td>108247269.7</td>
</tr>
<tr>
<td>read</td>
<td>2</td>
<td>0.940972</td>
<td>27888.85724</td>
</tr>
<tr>
<td>read</td>
<td>3</td>
<td>0.950511</td>
<td>27421.92881</td>
</tr>
</tbody>
</table>

See the following table for more details.

### Table 11-50 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Mode</td>
</tr>
<tr>
<td>Task</td>
<td>Task ID</td>
</tr>
<tr>
<td>Hit Rate</td>
<td>Cache hit rate</td>
</tr>
<tr>
<td>Throughput</td>
<td>Read/Write rate (MB/s)</td>
</tr>
</tbody>
</table>

### 11.6.3.4.14 RoCE Bandwidth

Obtain the `roce_{device_id}_{iter_id}.csv` file by referring to **11.6.3.4.1 Exporting Summary Reports**. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.
The file content is formatted as follows.

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>Time</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>Bandwidth (MB/s)</td>
</tr>
<tr>
<td>Rx Bandwidth efficiency</td>
<td>Receive bandwidth</td>
</tr>
<tr>
<td>rxPacket/s</td>
<td>Packet receive rate</td>
</tr>
<tr>
<td>rxError rate</td>
<td>Received packet error rate</td>
</tr>
<tr>
<td>rxDropped rate</td>
<td>Received packet loss rate</td>
</tr>
<tr>
<td>Tx Bandwidth efficiency</td>
<td>Transmit bandwidth</td>
</tr>
<tr>
<td>txPacket/s</td>
<td>Packet transmit rate</td>
</tr>
<tr>
<td>txError rate</td>
<td>Transmitted packet error rate</td>
</tr>
<tr>
<td>txDropped rate</td>
<td>Transmitted packet loss rate</td>
</tr>
<tr>
<td>funcId</td>
<td>RoCE ID</td>
</tr>
</tbody>
</table>
```

See the following table for more details.

### Table 11-51 Field description

11.6.3.15 AI CPU Top Functions

Obtain the `ai_cpu_top_function_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

```
<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>Cycles</th>
<th>Cycles (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>unknown</td>
<td>unknown</td>
<td>3.72E+09</td>
<td>85, 236318</td>
</tr>
<tr>
<td></td>
<td>/usr/lib/5.96E+08</td>
<td>13, 38198</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/adds</td>
<td>7.14E03</td>
<td>1, 599172</td>
</tr>
<tr>
<td></td>
<td>/usr/lib/17144485</td>
<td>0.383881</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/usr/lib/15383929</td>
<td>0.344403</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/usr/lib/14240949</td>
<td>0.313922</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/usr/lib/13102964</td>
<td>0.270955</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/klc</td>
<td>103590964</td>
<td>0.232738</td>
</tr>
<tr>
<td></td>
<td>/usr/lib/10342745</td>
<td>0.231611</td>
<td></td>
</tr>
</tbody>
</table>
```

See the following table for more details.

### Table 11-52 Field description
### 11.6.3.4.16 AI CPU PMU Events

Obtain the `ai_cpu_pmu_events_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Event</th>
<th>Name</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x11</td>
<td>Cpu_cycle</td>
<td>4.47E+09</td>
</tr>
<tr>
<td>0x8</td>
<td>Inst_ret</td>
<td>1.88E+09</td>
</tr>
</tbody>
</table>

See the following table for more details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Register value</td>
</tr>
<tr>
<td>Name</td>
<td>Event name</td>
</tr>
<tr>
<td>Count</td>
<td>Count value</td>
</tr>
</tbody>
</table>

### 11.6.3.4.17 Ctrl CPU Top Functions

Obtain the `ctrl_cpu_top_function_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>Cycles</th>
<th>Cycles(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>unknown</td>
<td>unknown</td>
<td>3.72E+09</td>
<td>85.2044</td>
</tr>
<tr>
<td>unknown</td>
<td>/usr/lib</td>
<td>5.88E+08</td>
<td>13.38541</td>
</tr>
<tr>
<td>unknown</td>
<td>/var/adds</td>
<td>7.1418926</td>
<td>1.599041</td>
</tr>
<tr>
<td>unknown</td>
<td>/usr/lib</td>
<td>1.7144608</td>
<td>0.38321</td>
</tr>
<tr>
<td>unknown</td>
<td>/usr/lib</td>
<td>1.3833329</td>
<td>0.244505</td>
</tr>
<tr>
<td>unknown</td>
<td>/usr/lib</td>
<td>1.42340083</td>
<td>0.318957</td>
</tr>
<tr>
<td>unknown</td>
<td>/usr/lib</td>
<td>1.2102634</td>
<td>0.271024</td>
</tr>
<tr>
<td>unknown</td>
<td>/var/shl</td>
<td>10.829965</td>
<td>0.233805</td>
</tr>
<tr>
<td>unknown</td>
<td>/usr/lib</td>
<td>1.90439745</td>
<td>0.231439</td>
</tr>
</tbody>
</table>

See the following table for more details.
Table 11-54 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Top functions of Ctrl CPU</td>
</tr>
<tr>
<td>Module</td>
<td>Module where the function is located</td>
</tr>
<tr>
<td>Cycles</td>
<td>Cycles taken to execute a function in the sampling period</td>
</tr>
<tr>
<td>Cycles(%)</td>
<td>Percentage of cycles taken to execute a function in the sampling period</td>
</tr>
</tbody>
</table>

11.6.3.4.18 Ctrl CPU PMU Events

Obtain the `ctrl_cpu_pmu_events_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Event</th>
<th>Name</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x11</td>
<td>Cpu_cycle</td>
<td>4.47E+09</td>
</tr>
<tr>
<td>0x8</td>
<td>Inst_ret</td>
<td>1.88E+09</td>
</tr>
</tbody>
</table>

See the following table for more details.

Table 11-55 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Register value</td>
</tr>
<tr>
<td>Name</td>
<td>Event name</td>
</tr>
<tr>
<td>Count</td>
<td>Count value</td>
</tr>
</tbody>
</table>

11.6.3.4.19 TS CPU Top Functions

Obtain the `ts_cpu_top_function_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Function</th>
<th>Cycles</th>
<th>Cycles(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x6ffffff04</td>
<td>727639</td>
<td>52.36656</td>
</tr>
<tr>
<td>0x6ffffff04</td>
<td>190000</td>
<td>17.85177</td>
</tr>
<tr>
<td>0x6ffffff04</td>
<td>108881</td>
<td>10.2131</td>
</tr>
<tr>
<td>0x6ffffff04</td>
<td>38000</td>
<td>3.570355</td>
</tr>
</tbody>
</table>

See the following table for more details.
## 11.6.3.4.20 TS CPU PMU Events

Obtain the `ts_cpu_pmu_events_{device_id}_{iter_id}.csv` file by referring to 11.6.3.4.1 Exporting Summary Reports. Replace `{device_id}` with the device ID and `{iter_id}` with the iteration ID.

The file content is formatted as follows.

<table>
<thead>
<tr>
<th>Event</th>
<th>Name</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3</td>
<td>Inst_ret</td>
<td>110340</td>
</tr>
<tr>
<td>0xa1</td>
<td>Cpu_cycle</td>
<td>1064529</td>
</tr>
</tbody>
</table>

See the following table for more details.

### Table 11-57 Field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Register value</td>
</tr>
<tr>
<td>Name</td>
<td>Event name</td>
</tr>
<tr>
<td>Count</td>
<td>Count value</td>
</tr>
</tbody>
</table>

## 11.7 Analyzing Profile Data

### 11.7.1 Overview

This section uses the MobileNetV2 and AlexNet models as examples to describe how to analyze the parsed profile data. The training performance of the MobileNetV2 model is better than that of the AlexNet model. This section mainly analyzes specific data (iteration and operator durations) collected in the training process to find performance bottlenecks on the current network for optimization.

**NOTE**

The data collected in this section may be different from the actual data. This is determined by the software version and does not affect the profile data analysis.
11.7.2 Analyzing Profile Data Collected During MobileNetV2 Training

For details about how to collect profile data during model training, see 11.5.1 Job Profiling. For details about how to parse the profile data, see 11.6.2.3 Obtaining Timeline Reports and 11.6.2.4 Obtaining Summary Reports.

11.7.2.1 Analyzing Iteration Durations

You can obtain the iteration track data file `training_trace_{device_id}_{iter_id}.json` by parsing the profile data. Enter `chrome://tracing` in the address box of Google Chrome, drag `training_trace_{device_id}_{iter_id}.json` to the blank area, and press the shortcut keys (w: zoom in; s: zoom out; a: move left; d: move right) on the keyboard to view the time consumed by each iteration. The following figure shows the timeline data.

![Figure 11-3 Total iteration duration](image)

As shown in Figure 11-3, the total duration of the second round of iterative training is 106.9 ms, in which the FP+BP duration is 96.6 ms. The FP+BP duration
(obtain it from Figure 11-4) accounts for 90% of the total duration. The gap 
(blank part in front of the FP+BP bar in Figure 11-4) is the data augmentation 
duration (mainly the time required for reading/loading data). The summary file 
corresponding to training_trace_{device_id}_{iter_id}.json is 
training_trace_{device_id}_{iter_id}.csv. Open the file. Then you can obtain 
formula Iteration Time = FP to BP Time + Grad Refresh + Data Aug, as shown 
in the following figure.

According to the preceding data, the time required for data augmentation (Data 
Aug) is much shorter than that for operator FP+BP. Therefore, data augmentation 
is not a performance bottleneck.

11.7.2.2 Analyzing Operator Durations

By parsing the profile data, you can obtain file 
op_summary_{device_id}_{iter_id}.csv that records the AI Core data, and file 
op_statistic_{device_id}_{iter_id}.csv that records the AI Core operator call 
quantity and time consumed.

- **op_summary_{device_id}_{iter_id}.csv**: This file collects statistics on the time 
  consumed by operators on the entire "bp_point + fp_point" link in a specific 
  iteration (first iteration by default). That is, the time consumed by all nodes 
  from bp_point to fp_point in the configuration parameters. In this file, pay 
  attention to the Task Duration column, which records the time consumed by 
  each operator. You can select Task Duration as the keyword to sort the table 
in descending order. The following figure shows the first part of the table. 
Among operators involved in the network, forward and backward operators of 
DepthwiseConv2D consume longer time. The time consumed by 
DepthwiseConv2DBackpropFilterD is the longest, that is, approximately 7 ms. 
Pay special attention to operators consuming milliseconds.

- **op_statistic_{device_id}_{iter_id}.csv**: This file collects statistics on operators 
  on the entire "bp_point + fp_point" link by type, regardless of the operator 
  names. For example, statistics of Conv2D operators are displayed in one row, 
  including the number of calls, total duration, average duration, maximum 
  duration, and minimum duration.
According to the preceding data, among the 2,590 operators, only one operator consumes 7 ms, 5 operators consume 2 and 3 ms, and 10 operators consume 1 to 2 ms. Operators with high time consumptions account for a small proportion.

11.7.3 Analyzing Profile Data Collected During AlexNet Training

For details about how to collect profile data during model training, see 11.5.1 Job Profiling. For details about how to parse the profile data, see 11.6.2.3 Obtaining Timeline Reports and 11.6.2.4 Obtaining Summary Reports.

11.7.3.1 Analyzing Iteration Durations

You can obtain the iteration track data file `training_trace_{device_id}_{iter_id}.json` by parsing the profile data. Enter `chrome://tracing` in the address box of Google Chrome, drag `training_trace_{device_id}_{iter_id}.json` to the blank area, and press the shortcut keys (w: zoom in; s: zoom out; a: move left; d: move right) on the keyboard to view the time consumed by each iteration. The following figure shows the timeline data.

**Figure 11-5** Total iteration duration
As shown in Figure 11-3, the total duration of the second round of iterative training is 131.4 ms, in which the FP+BP duration is 21.6 ms. The FP+BP duration (obtain it from Figure 11-4) accounts for 16.4% of the total duration. The gap (blank part in front of the FP+BP bar in Figure 11-4) is the data augmentation duration (mainly the time required for reading and loading data), which is about 108 ms. The summary file corresponding to training_trace_{device_id}_{iter_id}.json is training_trace_{device_id}_{iter_id}.csv. Open the file. Then you can obtain formula \(\text{Iteration Time} = \text{FP to BP Time} + \text{Grad Refresh} + \text{Data Aug}\), as shown in the following figure.

According to the preceding data, the time required for data augmentation (Data Aug) is 5 times that for operator FP+BP. Therefore, data augmentation is a performance bottleneck. Developers can optimize data loading and reading by improving the training code. For details, see “Training Performance Improvement > Data Preprocessing Performance Improvement” in the TensorFlow Network Model Porting and Training.

11.7.3.2 Analyzing Operator Durations

By parsing the profile data, you can obtain file op_summary_{device_id}_{iter_id}.csv that records the AI Core data, and file op_statistic_{device_id}_{iter_id}.csv that records the AI Core operator call quantity and time consumed.

- op_summary_{device_id}_{iter_id}.csv: This file collects statistics on the time consumed by operators on the entire “bp_point + fp_point” link in a specific iteration (first iteration by default). That is, the time consumed by all nodes from bp_point to fp_point in the configuration parameters. In this file, pay attention to the Task Duration column, which records the time consumed by each operator. You can select Task Duration as the keyword to sort the table.
in descending order. The following figure shows the first part of the table. Among operators involved in the network, MatMulV2 operators consume longer time. The longest time is approximately 5 ms. Pay special attention to operators consuming milliseconds.

![Table](image)

- **op_statistic_{device_id}_{iter_id}.csv**: This file collects statistics on operators on the entire "bp_point + fp_point" link by type, regardless of the operator names. For example, statistics of Mul operators are displayed in one row, including the number of calls, total duration, average duration, maximum duration, and minimum duration.

According to the preceding data, among the 182 operators, one operator consumes about 5 ms and 5 operators consume > 1 ms (15 ms in total). The time consumed by the 5 operators accounts for a high proportion. In addition, the 5 operators are of the same type MatMul, so you may need to optimize the operator performance.

## 11.8 Appendix

### 11.8.1 Profiling Options

**Description**

Sets Profiling options.

- **output**: path for storing profiling result files. Create the specified path in advance in the environment (either in a container or on the host) where training is performed. The running user configured during installation must have the read and write permissions on this path. The path can be an absolute path or a path relative to the path where the training script is executed.
  - An absolute path starts with a slash (/), for example, `/home/HwHiAiUser/output`.
  - A relative path starts with a directory name, for example, `output`.

- **training_trace**: iteration tracing switch. Collects software profile data of a training job and the AI Software Stack to profile the training job. Focuses on data augmentation, forward and backward propagation, and gradient
aggregation and update. When profiling mode is enabled to collect data generated in job profiling, `training_trace` must be set to **on**.

- **task_trace**: task tracing switch. Collects the HWTS hardware information of the Ascend AI Processor and the start and end of each task. Either **on** or **off**. A value other than **on** or **off** is equivalent to **off**. If `task_trace` is set to **on**, `training_trace` must also be set to **on**.

- **aicpu**: AI CPU data augmentation profiling switch. Either **on** or **off**. A value other than **on** or **off** is equivalent to **off**.

- **fp_point**: start point of the forward propagated operator in iteration traces, recording the start timestamp of forward propagation. Set the value to the name of the top operator in forward propagation. You can save the graph as a .pbtxt file by using `tf.io.write_graph` in the training script to obtain this name.

- **bp_point**: end point of the backward propagated operator in iteration traces, recording the end timestamp of backward propagation. `BP_POINT` and `FP_POINT` are used to compute the time used by forward and backward propagation. Set the value to the name of the bottom operator in backward propagation. You can save the graph as a .pbtxt file by using `tf.io.write_graph` in the training script to obtain this name.

- **aic_metrics**: AI Core metrics to profile.
  
  - **ArithmeticUtilization**: percentages of arithmetic utilization.
  
  - **PipeUtilization** (default): percentages of time taken by the compute units and MTEs.
  
  - **Memory**: percentages of external memory read/write instructions.
  
  - **MemoryL0**: percentages of internal memory read/write instructions.
  
  - **ResourceConflictRatio**: percentages of pipeline queue instructions.

  **NOTE**
  
  - `fp_point` and `bp_point` require manual configuration only in the dynamic shape scenario.
  
  - Online inference supports `task_trace` and `aicpu` but does not support `training_trace`.

### 11.8.2 AI Core Metrics

**NOTE**

The analysis result of AI Core metrics varies according to the AI Core profiling mode. This section assumes that the AI Core profiling mode is set to task-based.

- In task-based mode, profile data is collected task by task, and the profiling result is displayed in percentage of cycles.

- In sample-based mode, profile data is collected at a fixed interval, and the profiling result is displayed in percentage of time.

The analysis result of AI Core metrics is described as follows:

- **ArithmeticUtilization**
  
  - **mac_fp16_ratio**: percentage of cycles taken to execute Cube fp16 instructions
  
  - **mac_int8_ratio**: percentage of cycles taken to execute Cube int8 instructions
- **vec_fp32_ratio**: percentage of cycles taken to execute Vector fp32 instructions
- **vec_fp16_ratio**: percentage of cycles taken to execute Vector fp16 instructions
- **vec_int32_ratio**: percentage of cycles taken to execute Vector int32 instructions
- **vec_misc_ratio**: percentage of cycles taken to execute Vector misc instructions

*PipeUtilization*
- **vec_ratio**: percentage of cycles taken to execute Vector instructions
- **mac_ratio**: percentage of cycles taken to execute Cube instructions
- **scalar_ratio**: percentage of cycles taken to execute Scalar instructions
- **mte1_ratio**: percentage of cycles taken to execute MTE1 instructions (L1-to-L0A/L0B movement)
- **mte2_ratio**: percentage of cycles taken to execute MTE2 instructions (DDR-to-AI Core movement)
- **mte3_ratio**: percentage of cycles taken to execute MTE3 instructions (AI Core-to-DDR movement)
- **icache_miss_rate**: I-Cache miss rate

*Memory*
- **ub_read_bw**: UB read bandwidth (GB/s)
- **ub_write_bw**: UB write bandwidth (GB/s)
- **l1_read_bw**: L1 read bandwidth (GB/s)
- **l1_write_bw**: L1 write bandwidth (GB/s)
- **main_mem_read_bw**: main memory read bandwidth (GB/s)
- **main_mem_write_bw**: main memory write bandwidth (GB/s)

*MemoryL0*
- **l0a_read_bw**: L0A read bandwidth (GB/s)
- **l0a_write_bw**: L0A write bandwidth (GB/s)
- **l0b_read_bw**: L0B read bandwidth (GB/s)
- **l0b_write_bw**: L0B write bandwidth (GB/s)
- **l0c_read_bw**: L0C read bandwidth (GB/s)
- **l0c_write_bw**: L0C write bandwidth (GB/s)

*ResourceConflictRatio*
- **vec_bankgroup_cflt_ratio**: percentage of cycles taken to execute vec_bankgroup_stall_cycles instructions
- **vec_bank_cflt_ratio**: percentage of cycles taken to execute vec_bank_stall_cycles instructions
- **vec_resc_cflt_ratio**: percentage of cycles taken to execute vec_resc_cflt_ratio instructions
11.8.3 Installing perf, iotop, and ltrace

Installing Dependencies

NOTE
This section describes how to install the perf, iotop, and ltrace tools in Ubuntu 18.04 and CentOS 7.6.

- The installation command applies to Ubuntu 18.04.
  
  \texttt{apt-get install perf iotop ltrace}

- The installation command applies to CentOS 7.6.
  
  \texttt{yum install perf iotop ltrace}

Configuring User Permissions

After installation, you need to set user permissions on the dependencies. The following uses the user process 12345 as an example.

Step 1  Log in to the environment as the root user.

Step 2  Open the /etc/sudoers file.

   \texttt{chmod u+w /etc/sudoers}

   \texttt{vi /etc/sudoers}

Step 3  Add the following lines to the file.

\begin{verbatim}
username  ALL=(ALL:ALL) NOPASSWD:/usr/bin/perf trace -T --syscalls -p 12345,
/usr/bin/ltrace -ttt -T -f -e pthread_* -p 12345
/usr/sbin/iotop -b -d 0.02 -p 12345 -P -t
/usr/bin/ltrace --version
/usr/bin/perf --version
/usr/bin/iotop --version
/usr/bin/pkill -2 ltrace
/usr/bin/pkill -2 iotop
/usr/bin/pkill -2 perf
\end{verbatim}

   In the preceding command, \texttt{username} indicates the user name, and \texttt{12345} indicates the ID of the process started by the user. You can modify the configuration based on the trustlist. For security considerations, you are not advised to use wildcards to configure process IDs.

Step 4  Run the \texttt{:wq!} command to save the file.

Step 5  Remove the write permission on the /etc/sudoers file.

   \texttt{chmod u-w /etc/sudoers}

11.8.4 FAQs

11.8.4.1 Profiling Fails Due to Full Disk Space

Symptom

The error message shown in Figure 11-7 is displayed when a training job is delivered in system profiling or job profiling.
Figure 11-7 Error message: No usable temporary directory

Possible Cause

No usable temporary directory is displayed, the system disk space maybe full.

Solution

Perform the following steps:

Step 1  Delete unnecessary files in the system disk.

Step 2  Run the `df -h` command to check whether the disk has available space.

----End
12 Auto Tune Instructions

12.1 About This Document

Purpose
This document provides instructions for using Auto Tune for operator tuning and basic troubleshooting techniques.

Intended Audience
This document is intended for developers who develop models based on the Ascend AI Processor. After reading this document, you will be able to:

Use the Auto Tune tool to find the optimal tiling policy of operators to meet the operator and network performance requirements.

12.2 Introduction to Auto Tune

12.2.1 Overview

Why Does Auto Tune Matter?
- To fully utilize the compute power of AI processors, the compute process needs to be well organized.

   An AI processor consists of multiple compute units, on-chip storage, data transfer module, and many other modules. The execution time of an operator...
running on the processor cannot be obtained by simply dividing the compute amount by the compute power. The collaboration situation between the components should also be considered. For a compute task deployed on the same compute processor, the compute efficiency varies greatly depending on the pipeline design. A small change to the compute inputs may require a brand-new pipeline design. Only well-designed scheduling logic can give full play to the compute power of the hardware backend.

- The pipeline, or the timing between compute components, must be well-designed to achieve optimal performance.

The theoretical peak performance of an operator is obtained by dividing the bottleneck load (including data compute and transfer) by the efficiency of the processing units. Due to limited on-chip storage, a compute task will be tiled prior to processing, which brings certain compute and transfer redundancy. Therefore, the actual load is usually greater than the theoretical load. The redundancy of a compute task varies depending on the pipeline design. Generally, a solution with low redundancy is selected or the redundancy is transferred to non-bottleneck components. Therefore, to achieve optimal performance, the timing between the components needs to be properly designed.

Figure 12-1 Efficiency comparison of different timings

- Operator scheduling is too complex to be covered by experience.

In TBE, the arrangement of compute components on a pipeline is basically determined by the Schedule module. TVM introduces a number of processing layers between schedule and hardware behavior to improve the applicability. Therefore, scheduling operations could vary significantly in terms of the hardware behavior, which means operator scheduling is too complex to be covered by experience.

The following table compares the time consumption and cost analysis between manual tuning and auto tuning.

<table>
<thead>
<tr>
<th></th>
<th>Time Consumption</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual tuning</td>
<td>Long, in the unit of days</td>
<td>Requires experienced experts.</td>
</tr>
<tr>
<td>Auto tuning</td>
<td>Short, in the unit of minutes</td>
<td>Mainly requires machine resources and reduces manual intervention.</td>
</tr>
</tbody>
</table>

Simply put, it is complex to maximize the operator performance on the the Ascend AI Processor. Using auto tuning, instead of manual tuning, is a much more expeditious way to explore hardware performance.
Auto Tune Functions

Auto Tune can automatically tune operators by leveraging hardware resources.

During the generation of your network model, you only need to enable `auto_tune_mode` to enable the Auto Tune tool. Then, this tool can be automatically called for operator tuning during operator building, and the tuning result is stored in the custom repository. After that, the operator can achieve the tuned performance when it is called again.

The current version of Auto Tune supports only the auto tuning of AI Core operators whose compute logic is implemented using DSL APIs. For details about the supported operators, see 12.5.1 Operator Lists.

12.2.2 Architecture

Tuning Modes

The Auto Tune tool supports the following two tuning modes:

- **Reinforcement Learning (RL)**
  
  Supports elewise, broadcast, and reduce operators. For details, see Operators Supporting RL Tuning.
  
  Abstracts the schedule as a decision chain based on Monte Carlo tree search (MCTS), a heuristic search algorithm for some kinds of decision processes, and then employs RL-trained Neural Networks (NNs) to guide decision-making.

- **Genetic Algorithm (GA)**
  
  Supports Cube operators. For details, see Operators Supporting GA Tuning.
  
  Generates a tuning space through multi-level combinations with manual experience for pruning and sorting, improving tuning efficiency, and employs the GA to tune parameters in rounds to obtain the optimal tiling policy.

RL Tuning

RL consists of two parts, RL Tuning and RL Repositories.

Figure 12-2 shows the RL architecture.

Figure 12-2 RL architecture

RL Tuning consists of the Controller and Environment.
**Controller** controls the operator, including the RL algorithm, MCTS, and Policy Model (optimal model evaluation criteria based on tuned operators). The following figure shows the principle of the Policy Model of RL tuning.

**Figure 12-3 RL Policy Model training**

![RL Policy Model training](image)

- a. Initialize the neural network.
- b. Perform sampling by using the neural network and MCTS to obtain the operator’s schedule policy.
- c. Build the operator obtained after sampling, evaluate the operator running performance on the board, and generate the reward to obtain a training sample.
- d. Train the neural network in a supervised manner.
- e. Repeat b to d until the network result is converged.

**Environment** is used to evaluate the tuning result. **Rules** refer to the schedule generation rules, and **Estimator** refers to the time estimation policy.

RL tuning provides a built-in repository and a custom repository.

- The built-in repository contains the preset tiling policies for frequently-used operators under different shapes. Find it in `opp/data/rl/<soc_version>/built-in` in the OPP installation path.
- The custom repository contains the tiling policies with better performance than the built-in repository generated after RL tuning. The default custom repository is stored in `data/rl/<soc_version>/custom` under the FwkACLlib installation directory. You can also specify an alternative.

**GA Tuning**

The GA tuning process is as follows: (1) analyze the tuning space; (2) perform tuning using the genetic algorithm; (3) obtain the optimal tiling policy and add it to the repository.

The following figure shows the GA architecture.
GA tuning provides a built-in repository and a custom repository.

- The built-in repository contains the preset tiling policies for frequently-used operators under different shapes. Find it in `opp/data/tiling/<soc_version>/built-in` in the OPP installation path.
- The custom repository contains the tiling policies with better performance than the built-in repository generated after GA tuning. The default custom repository is stored in `data/tiling/<soc_version>/custom` under the FwkACLlib installation directory.

The following figure shows the GA tuning principles.

**Figure 12-4 GA tuning principles**

12.2.3 Tuning Workflow

During the generation of a network model, the Auto Tune tool performs tuning during operator building. The following figure shows the default tuning workflow.
1. Import the third-party network model to the GE and FE for graph preparation (such as shape inference and operator selection) and graph tuning (such as fusion and constant folding).

2. Start operator building. The detailed building process is as follows:
   a. Look up tiling policy matches in existing repositories for each layer.
      - If hit:
          - If either Auto Tune or `REPEAT_TUNE` is disabled, build the operator using the tiling policy match.
          - If both Auto Tune and `REPEAT_TUNE` are enabled, perform tuning again.
            If the tiling policy after tuning is better than the existing policies in the built-in repository and custom repository, it is saved to the custom repository and is used to build the operator.
            Otherwise, no custom repository is generated, and the existing repository is directly used to build the operator.
      - If missed:
        Check if Auto Tune is enabled.
        - If Auto Tune is enabled:
If the tiling policy after tuning is better than the default tiling policy, it is saved to the custom repository and is used to build the operator.
Otherwise, the default tiling policy is stored in the custom repository and the custom repository is used to build the operator.
  ○ If Auto Tune is not enabled, the default tiling policy is used to build the operator.

3. In the training scenario, a trained model is generated after build.

12.3 Tool Usage

12.3.1 Auto Tuning in Training Scenario

12.3.1.1 Environment Setup

- The Auto Tune tool runs on the Ascend AI Processor. Currently, this tool supports only the development+commissioning environment, that is, the Ascend-CANN-Toolkit.

⚠️ NOTE

- Set up the environment by referring to CANN Software Installation Guide.
- This tool supports Python 3.7.0 to Python 3.7.9. This document uses Python 3.7.5 as an example. The environment variables and installation commands vary according to the actual Python version.
- Ensure that the available disk space in the home directory of the user who performs tuning in the operating environment is at least 20 GB.
- The Auto Tune tool is stored in the python/site-packages/schedule_search and python/site-packages/auto_tune directories under the FwkACLlib installation path.

- Third-party software

After the environment is deployed, install the third-party software that the Auto Tune tool depends on. For details, see Table 12-1.

<table>
<thead>
<tr>
<th>Third-Party Software</th>
<th>Description</th>
<th>Installation Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow 1.15</td>
<td>For guiding operator search in RL tuning.</td>
<td>For the installation procedure, see &quot;Installing the Deep Learning Framework &gt; Installing TensorFlow 1.15.0&quot; in CANN Software Installation Guide.</td>
</tr>
</tbody>
</table>
### 12.3.1.2 Environment Variable Configuration

Before starting the Auto Tune tool, run the `export` command to declare environment variables on the terminal. The declared environment variables become invalid when the Shell terminal is closed.

The following is an example of the environment variable configuration.

```bash
export install_path=/home/HwHiAiUser/Ascend/ascend-toolkit/latest
export PATH=/usr/local/python3.7.5/bin:${install_path}/fwkacllib/ccec_compiler/bin:${install_path}/fwkacllib/bin::PATH # Specify the Python 3.7.5 installation path if multiple Python 3 versions exist.
export LD_LIBRARY_PATH=${install_path}/fwkacllib/lib64:$LD_LIBRARY_PATH
export ASjudul DEVICE_ID=0
export TUNE_BANK_PATH=/home/HwHiAiUser/custom_tune_bank
export REPEAT_TUNE=False
export TUNE_OPS_NAME=conv_layers/Pad_1 # Names of the nodes to be tuned. If specified, only the specified nodes are tuned.
export ASCEND_OPP_PATH=${install_path}/opp
# Optional environment variables of Auto Tune
export TUNE_BANK_PATH=/home/HwHiAiUser/custom_tune_bank
export REPEAT_TUNE=False
export TUNE_OPS_NAME=conv_layers/Pad_1 # Names of the nodes to be tuned. If specified, only the specified nodes are tuned.
export ASCEND_DEVICE_ID=0
export TUNE_BANK_PATH=/home/HwHiAiUser/custom_tune_bank
export ENABLE_TUNE_BANK=True
# Environment variable for offline tuning
export ENABLE_TUNE_DUMP=True
# Optional environment variables for offline tuning
export TUNE_DUMP_PATH=/home/HwHiAiUser/DumpData
```

### NOTE

- **install_path** indicates the FwkACLlib and OPP installation path. Replace it with the actual path.
- You can save the commands for setting environment variables to the operator custom script for future use.
The following table describes the environment variables.

**Table 12-2 Description of environment variables**

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| LD_LIBRARY_PATH          | (Required) Dynamic library path. Set this variable according to the preceding example.  
                         | *$\{install\textunderscore path\}/fwkacllib/lib64*: required for executing the training script.                                           |
| PATH                     | (Required) Executable program path. Set this variable by referring to the preceding example.  
                         | - *$\{install\textunderscore path\}/fwkacllib/ccec Compiler/bin*: required for TBE operator build.                                        |
|                          | - *$\{install\textunderscore path\}/fwkacllib/bin*: required by the GA algorithm of Auto Tune.                                                |
| PYTHONPATH               | (Required) Python search path.                                                                                                                                 |
| ASCEND_OPP_PATH          | (Required) OPP root directory. Set this variable according to the preceding example.  
                         | It is an environment variable required for operator build.                                                                                        |
| ASCEND_DEVICE_ID         | (Optional) Logical ID of the Ascend AI Processor.                                                                                             |
|                          | The value range is \([0, N - 1]\) and the default value is 0. \(N\) indicates the device count in the physical machine, VM, or container. |
| TE_PARALLEL_COMPILER     | (Optional) Parallel build enable.                                                                                                             |
|                          | Parallel build is especially useful when a deep network is to build.  
                         | **TE\_PARALLEL\_COMPILER** indicates the number of parallel operator build processes.  
<pre><code>                     | The value must be an integer and defaults to 8. When the value is greater than 1, parallel build is enabled.                                                                 |
</code></pre>
<p>|                          | The value range is ([1, 32]). The maximum value is calculated as follows: Number of CPU cores * 80% / Number of Ascend AI Processors. |</p>
<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| TUNE_BANK_PATH       | (Optional) Path of the custom repository generated after Auto Tune. 

For **Operators Supporting RL Tuning:**
- If this environment variable is not set, the custom repository is stored in the `fwkacllib/data/rl/<soc_version>/custom` directory of the FwkACLlib installation path by default.
- If this environment variable is configured, the optimal policy obtained after RL tuning is saved to the `<soc_version>/rl` directory of the configured path. The tuning script automatically creates the `<soc_version>/rl` directory in the path specified by `TUNE_BANK_PATH`.

The value can be an absolute path or a path relative to the Auto Tune path. The specified path must exist and the running user must have the read, write, and execute permissions on the specified path.

For example, if set to `TUNE_BANK_PATH=./custom_bank`, the optimal policy obtained after RL tuning is saved to the `/custom_bank/<soc_version>/rl` directory.

For **Operators Supporting GA Tuning:**
- If this environment variable is not set, the custom repository is stored in the `fwkacllib/data/tiling/<soc_version>/custom` directory of the FwkACLlib installation path by default.
- If this environment variable is configured, the optimal policy obtained after GA tuning is saved to the `<soc_version>/ga` directory of the configured path. The tuning script automatically creates the `<soc_version>/ga` directory in the path specified by `TUNE_BANK_PATH`.

Set this variable to an absolute path or a path relative to the path of the Auto Tune tool. The specified path must exist and the running user must have the read, write, and execute permissions on the specified path.

For example, if set to `TUNE_BANK_PATH=./custom_bank`, the
optimal policy obtained after GA tuning is saved to the `/custom_bank/<soc_version>/ga` directory.

**NOTE**
If the repository path is customized before tuning, you also need to configure this environment variable if you want to use the custom repository during model conversion.

For example, if the custom repository is stored in the `/home/HwHiAiUser/custom_bank/<soc_version>/ga` directory, you need to configure the following environment variables when running the training script:

```bash
export TUNE_BANK_PATH=/home/HwHiAiUser/custom_bank
```

### REPEAT_TUNE

(Optional)
Repeat tuning enable.

If set to `False` and a network tuning case is available in the repository (built-in or custom), the tuning process of the case is skipped. When the logic of some operators is changed, for example, the ND input support is added to the GEMM operator. In this case, you need to set this environment variable to `True` and initiate tuning again.

Value range: either `True` or `False`. Defaults to `False`.

### ENABLE_TUNE_BANK

(Optional)
Repository enable during operator build.

- `True`: enabled. During operator build, the tiling policy in the repository is automatically obtained. The custom repository has higher priority over the built-in repository.
- `False`: disabled.

Defaults to `True`.

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>optimal policy obtained after GA tuning is saved to the <code>/custom_bank/&lt;soc_version&gt;/ga</code> directory.</td>
</tr>
<tr>
<td><strong>NOTE</strong></td>
<td>If the repository path is customized before tuning, you also need to configure this environment variable if you want to use the custom repository during model conversion. For example, if the custom repository is stored in the <code>/home/HwHiAiUser/custom_bank/&lt;soc_version&gt;/ga</code> directory, you need to configure the following environment variables when running the training script: <code>export TUNE_BANK_PATH=/home/HwHiAiUser/custom_bank</code></td>
</tr>
<tr>
<td><strong>REPEAT_TUNE</strong></td>
<td>(Optional) Repeat tuning enable. If set to <code>False</code> and a network tuning case is available in the repository (built-in or custom), the tuning process of the case is skipped. When the logic of some operators is changed, for example, the ND input support is added to the GEMM operator. In this case, you need to set this environment variable to <code>True</code> and initiate tuning again. Value range: either <code>True</code> or <code>False</code>. Defaults to <code>False</code>.</td>
</tr>
<tr>
<td><strong>ENABLE_TUNE_BANK</strong></td>
<td>(Optional) Repository enable during operator build. <code>True</code>: enabled. During operator build, the tiling policy in the repository is automatically obtained. The custom repository has higher priority over the built-in repository. <code>False</code>: disabled. Defaults to <code>True</code>.</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TUNE_OPS_NAME</td>
<td>(Optional) Specified-layer tuning, which is used in network tuning scenarios. For example, after profiling a network, you can use this environment variable to tune particular operator (or operators separated by commas) with low performance.</td>
</tr>
<tr>
<td></td>
<td>- Node (or nodes separated by commas) to tune on a network. The node names must be the <strong>OP Names</strong> of the nodes on the network that is processed by the GE and FE and adapts to the Ascend AI Processor. The <strong>OP Names</strong> can be obtained from the profile data. For details, see <a href="#">Profiling Instructions</a>. You can search for the keyword SelectTuneMode in the log generated during model generation to check whether the specified nodes are tuned.</td>
</tr>
<tr>
<td></td>
<td>- If this environment variable is not specified, all supported operators on the network, as listed in <a href="#">12.5.1 Operator Lists</a>, will be tuned.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong> This environment variable applies to tuning along with network model generation only and does not support offline tuning based on dump data.</td>
</tr>
<tr>
<td>enables_tune_dump</td>
<td>(Optional) Applies to <strong>offline tuning based on dump data</strong>. Operator dump enable for offline tuning. Value range: either <strong>True</strong> or <strong>False</strong>. Defaults to <strong>False</strong>.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong> If this environment variable is set to <strong>True</strong>, online tuning is not performed even if Auto Tune is enabled (only dump data is generated).</td>
</tr>
</tbody>
</table>
### Environment Variable Configuration

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| TUNE_DUMP_PATH       | (Optional)  
Applies to **offline tuning based on dump data**. 
Dump path for offline tuning.  
Set the environment variable to an absolute path or a path relative to the location of Auto Tune. You can specify a path that is readable, writable, and executable for any user.  
If the environment variable is not configured, the **tune_dump** directory will be generated in the tool execution path for storage by default. |

### 12.3.1.3 Tuning Procedure

#### Prerequisites

- Prepare the development environment and operating environment by referring to **12.3.1.1 Environment Setup** and install the required software.
- Configure the environment variables on which the Auto Tune tool depends by referring to **12.3.1.2 Environment Variable Configuration**.

#### Online Tuning

In the training script, enable Auto Tune in either of the following ways.

- If the **initialize_system** API is called in the training script, enable Auto Tune as follows:
  ```python
  npu_init = npu_ops.initialize_system()  
npu_shutdown = npu_ops.shutdown_system()  
config = tf.ConfigProto()  

custom_op.parameter_map["auto_tune_mode"]= tf.compat.as_bytes("RL,GA")  
...
with tf.Session(config=config) as sess:
    sess.run(npu_init)  
    # Call the HCCL API...
    # Perform training...
    sess.run(npu_shutdown)
  ```

- If the **initialize_system** API is not called in the training script, enable Auto Tune as follows:
  - Set the **auto_tune_mode** parameter in the session configuration option:
    ```python
    session_config = tf.ConfigProto()  
custom_op = session_config.graph_options.rewrite_options.custom_optimizers.add()  
...
    custom_op.parameter_map["auto_tune_mode"]= tf.compat.as_bytes("RL,GA")
    ```
  - Set the **auto_tune_mode** parameter in NPURunConfig:
    ```python
    session_config=tf.ConfigProto()  
config = NPURunConfig(auto_tune_mode="RL,GA", session_config=session_config)
    ```

**auto_tune_mode** can be set to any of the following values:
● "RL,GA": Both RL and GA are used for tuning. The sequence of RL and GA is not sensitive. The Auto Tune tool automatically selects the RL mode or GA mode according to the operator characteristics.

● "RL": Only Operators Supporting RL Tuning are tuned.

● "GA": Only Operators Supporting GA Tuning are tuned.

For details about other configurations in the training script, see TensorFlow Network Model Porting and Training.

- To generate INFO-level Auto Tune log to a host log file, set the host log level to INFO and run the training script again. For details about log level settings, see Log Reference.

- You can set the following tuning functions through environment variables:
  - If an operator in the network model has a match in the repository, the operator will not be tuned repeatedly by default. You can configure the REPEAT_TUNE environment variable to forcibly tune the operator again.
  - You can configure the TUNE_OPS_NAME environment variable to tune a specified operator layer.

---

**NOTICE**

- The Auto Tune tool also provides other environment variable functions. For details, see 12.3.1.2 Environment Variable Configuration.

- If the multi-device training script is used to start multiple training processes with Auto Tune enabled, the same operators are tuned and the same repositories are used across the training processes, which does not improve the tuning efficiency. In this scenario, HCCL connection setup may time out due to the tuning time difference across the training processes.

Therefore, in the multi-device training scenario, you need to perform tuning on each device separately before you start multi-device training.

- It is allowed to start multiple training processes for tuning on the host. Proper process parallelism improves the tuning efficiency. However, due to resource restrictions, the tuning efficiency decreases when the number of parallel processes reaches a certain limit. The following condition should be met:
  
  Training process count * TE_PARALLEL_COMPILER * 2 < Host CPU core count, where TE_PARALLEL_COMPILER indicates the number of parallel operator build processes.

  In the TBE operator parallel build scenario (that is, TE_PARALLEL_COMPILER > 1), it is recommended that one training process correspond to one device.

- The owner group of the custom repository generated after tuning is the user who starts tuning, with permission 640. Users outside the owner group (except the super user) do not have the permission. If you want other users to access this custom repository, manually assign the permission to these users.
Offline Tuning Based on Dump Data

The Auto Tune tool supports offline tuning based on the operator output description. In offline tuning, Auto Tune is decoupled from the online training job. To start offline tuning with Auto Tune, perform the following steps:

- Obtain the dump data (including the operator output description file and operator binary file) when running the training script to generate a network model.
- Execute the Auto Tune tool to perform offline tuning with the obtained dump data.

The detailed operations are as follows:

**Step 1** Obtain the dump data when running the training script to generate a network model.

The dump data refers to the operator output description file and the operator binary file. The prerequisites for generating the dump data include:

   - **LD_LIBRARY_PATH**, **PYTHONPATH**, and **ASCEND_OPP_PATH** are required environment variables for configuring Auto Tune. For details, see 12.3.1.2 Environment Variable Configuration.
     ```
     export install_path=/home/HwHiAiUser/Ascend/ascend-toolkit/latest
     export LD_LIBRARY_PATH=${install_path}/fwkacllib/lib64:$LD_LIBRARY_PATH
     export PYTHONPATH=${install_path}/fwkacllib/python/site-packages:$PYTHONPATH
     export ASCEND_OPP_PATH=${install_path}/opp
     ```
   - Enable dump.
     ```
     export ENABLE_TUNE_DUMP=True
     ```
   - Set the dump path.
     ```
     export TUNE_DUMP_PATH=/home/HwHiAiUser/DumpData
     ```
2. Run the training script to generate a network model (Auto Tune does not need to be enabled) and generate dump data.
   
   For details, see TensorFlow Network Model Porting and Training.
   
   After the training script is executed, dump data is generated in the path specified by **TUNE_DUMP_PATH**.

**Step 2** Perform offline tuning based on the dump data with Auto Tune.

The entry script for offline tuning is in **fwkacllib/python/site-packages/schedule_search.egg/schedule_search/msoptune.py** under the FwkACLlib installation directory. Run this Python script to start offline tuning with the following command:

```python
python3.7 {msoptune.py path} --start {dump path}
```

Example:

```python
python3.7 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/fwkacllib/python/site-packages/schedule_search.egg/schedule_search/msoptune.py --start /home/HwHiAiUser/DumpData
```

- Replace `/home/HwHiAiUser/Ascend/ascend-toolkit/latest` with the actual FwkACLlib installation path.
- Replace `/home/HwHiAiUser/DumpData` with the actual dump path. It can be either an absolute path or a path relative to the directory where the current script is executed.
NOTICE

- Currently, only one process is allowed for offline tuning on the host.
- The owner group of the custom repository generated after tuning is the user who starts tuning, with permission 640. Users outside the owner group (except the super user) do not have the permission. If you want other users to access this custom repository, manually assign the permission to these users.

---End

12.3.1.4 Tuning Result

This topic describes the changes of the repositories and the tuning result file after tuning is complete.

Custom Repository

After the tuning is complete, if the conditions for generating a custom repository are met (see 12.2.3 Tuning Workflow), a custom repository is generated.

The custom repository is stored in the path specified by the TUNE_BANK_PATH environment variable. If this environment variable is not set:

- For Operators Supporting RL Tuning, the custom repository will be generated to `fwkacllib/data/rl/<soc_version>/custom/` in the FwkACLlib installation path.
- For Operators Supporting GA Tuning, the custom repository will be generated to `fwkacllib/data/tiling/<soc_version>/custom/` in the FwkACLlib installation path.

Tuning Result File

When the tuning starts, a file named `tune_result_{timestamp}_pidxxx.json` is generated in the Auto Tune working directory, which records the tuning process and result.

{timestamp} is in the format of YYYYMMDD_HHMMSSMS. xxx in pid.xxx indicates the process ID.

The following gives an example of the file.

```json
"[ 'Operator Name']":{
  "result_data":{
    "after_tune": 56,
    "before_tune": 66,
  }
  "status_data":{
    "bank_append": true,
    "bank_hit": false,
    "bank_reserved": false,
    "bank_update": false
  }
  "ticks_best":[
    [82, 2020-08-08 18:03:38],
    [104, 2020-08-08 18:03:50],
    ...
  ]
},
```
• **Operator Name**: name of the operator in the original graph. If graph fusion is performed during the tuning and the fused node comes from multiple nodes in the original graph, multiple operator names are displayed, for example:
  ```
  [['scale5a_branch1', 'bn5a_branch1', 'res5a_branch1'], ['res5a'], ['res5a_relu']]
  ```

• **result_data**: tuning result, including the time taken to execute the operator on the network before and after tuning.
  - **after_tune**: time (μs) taken to execute the operator after Auto Tune is performed.
  - **before_tune**: time (μs) taken to execute the operator before Auto Tune is performed.

• **status_data**: detailed tuning status of all operators on the network.
  - **bank_append**: If it is true, the tiling policy of the operator is appended to the repository after tuning (the repository before tuning contains no tiling policy of the operator); else, false.
  - **bank_hit**: If it is true, a tiling policy hit of the operator is found in the repository; else, false.
  - **bank_reserved**: If it is true, the tiling policy of the operator in the repository remains unchanged after tuning; else, false.
  - **bank_update**: If it is true, the tiling policy of the operator in the repository is updated after tuning; else, false.

• **ticks_best**: records the operator tuning result per iteration, including the tiling elapsed time and the tuning end time.

**NOTE**

- During the tuning, a `tune_show_{timestamp}_pidxxx` folder is generated in the tuning working directory, which stores the flag file of each tuned operator. If you want to cancel the tuning of an operator, run the following command:

  ```
  python3.7 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/fwkacllib/python/site-packages/schedule_search.egg/schedule_search/msoptune.py --stop tune_show_{timestamp}_pidxxx
  ```

  Select the operator that you want to cancel tuning as prompted. When tuning is complete, the tool compares the current repository with the existing repository. If the current repository outperforms the existing one, the existing repository is replaced in the custom directory or a new repository is added to the custom directory. Otherwise, no new repository is generated.

  The `tune_show_{timestamp}_pidxxx` folder is automatically deleted after tuning is complete.

- When tuning is complete, you can run the following command to generate the tuning status of all operators on the network based on the `tune_result_{timestamp}_pidxxx.json` file:

  ```
  python3.7 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/fwkacllib/python/site-packages/schedule_search.egg/schedule_search/msoptune.py --summary tune_result_{timestamp}_pidxxx.json
  ```

  In the command output, **Total_Num** indicates the total number of operators on the network after graph tuning and fusion, **Hit_Num** indicates the number of operators that have tiling policy hits in the repository, **Append_Num** indicates the number of operators with tiling policies appended to the repository after tuning, **Update_Num** indicates the number of operators whose tiling policies in the repository are updated after tuning, and **Reserved_Num** indicates the number of operators whose tiling policies in the repository remain unchanged.
12.3.2 Repository Merging

Overview

You can merge custom repositories by using the Auto Tune tool in different directories and save the tiling policies that are superior to those in the built-in repository to the output directory.

**NOTE**

Only custom repositories of the same the Ascend AI Processor model can be merged.

Procedure

**Step 1**

Copy the repositories to be merged generated in different environments to different directories in the target environment, for example, `{src_dir1}` and `{src_dir2}`. The repositories to be merged cannot be stored in the default custom repository directory.

- In RL tuning, the custom repository is stored in `/data/rl/{soc_version}/custom` under the FwkACLlib installation directory by default.
- In GA tuning, the custom repository is stored in `/data/tiling/{soc_version}/custom` under the FwkACLlib installation directory by default.

**Notes:**

The repositories to be merged must be stored in the `{soc_version}` directory in `{src_dir}`. You need to create a level-2 directory `{soc_version}`/rl or `{soc_version}`/ga in `{src_dir}` to store the RL or GA custom repository of the corresponding the Ascend AI Processor version. If the `{soc_version}`/rl or `{soc_version}`/ga directory already exists in `{src_dir}`, you do not need to create it again.

Replace `{soc_version}` with the the Ascend AI Processor version in use. It is the exact name of the .ini file in `data/platform_config` under the ATC or FwKACLlib installation directory.

If you still cannot determine the `{soc_version}` field using the preceding method, perform the following operations:

1. Click [here](#) to download CANN `{version}` Ascend-DMI Tool User Guide 01.
3. Run the related command to view the processor details. For example, in the output of the `ascend-dmi -i -dt` command, Chip Name field corresponds to `$soc_version`.

**NOTICE**

For Ascend 910 AI Processor, `$soc_version` is `Ascend910`.

**Step 2**

Merge the repositories.
1. Set the environment variables.
   Set the environment variables `LD_LIBRARY_PATH`, `PYTHONPATH`, and `ASCEND_OPP_PATH`.
   ```
   export install_path=/home/HwHiAiUser/Ascend/ascend-toolkit/latest
   export LD_LIBRARY_PATH=${install_path}/fwkacllib/lib64:$LD_LIBRARY_PATH
   export PYTHONPATH=${install_path}/fwkacllib/python/site-packages:$PYTHONPATH
   export ASCEND_OPP_PATH=${install_path}/opp
   ```

2. Merge the repositories.
   Run the following command:
   ```
   python3.7 /home/HwHiAiUser/Ascend/ascend-toolkit/latest/fwkacllib/python/site-packages/schedule_search.egg/schedule_search/msoptune.py --merge {src_dir1}:{src_dir2} {dst_dir}
   ```
   - `{src_dirx}`: Separate multiple `src_dir` directories by colons (:). `src_dir` is the `{src_dir}` directory specified in Step 1 for storing the custom repositories to be merged. It can be an absolute path or a relative path. For example, if the custom repository is stored in the `/home/HwHiAiUser/data/ascend310/ga` directory, set `src_dir` to `/home/HwHiAiUser/data`.
   - `dst_dir`: output directory of the merged custom repository. It can be an absolute path or a relative path. The path must exist and the `msoptune.py` script must have the read, write, and execute permissions on the path.

   The `msoptune.py --merge` script provides the following functions:
   - Merge the custom repositories in the `{src_dir}` directory.
   - Compare the merged custom repository with the built-in repository, and place the tiling policies that do not exist in the built-in repository or are better than the built-in repository into `{dst_dir}`.

   The merged RL and GA repositories in `{dst_dir}` are stored in `{soc_version}/rl` and `{soc_version}/ga`, respectively. The directories are automatically created with the script execution.

--- End

12.4 Troubleshooting

Common Errors in RL Tuning

- Error "unknown op compute."
  Check whether the operator is supported. Currently, only the elewise, broadcast, and reduce operators are supported. For details about all the supported operators, see 12.5.1 Operator Lists.

- Error "import base64 in python3.7 failed in host XX, please fix it!"
  An error is reported during the operating environment check, indicating that the base64 component is missing from Python 3.7. Ensure that the TBE development environment is ready before RL tuning.

  Run the following command to install the base64 component:
  ```
  pip3.7 install pybase64
  ```

- Error: "The avail space of /home/HwHiAiUser in XXX is smaller than 1G, please fix it!"
An error is reported during the operating environment check, indicating that the available space of the host is less than 1 GB. In this case, clear the space and try again.

- Error: "stage[xx] > max_stages[128]."
The number of stages of the current operator exceeds the allowed maximum 128. Tuning of the operator is not supported currently.

**Common Errors in GA Tuning**

- Error: "there is no kernel_perf_comm in PATH!"
Check that `${install_path}/atc/bin` or `${install_path}/fwkacllib/bin` has been added to the environment variable.

- Error: "Failed run kernel too many!"
The following message is displayed in the tuning log:

```
kernelName:xxxx, ResultStatus:0-255, TotalCycle:0-xxx
```

**KernelName** is the name of the current `.o` file.

**ResultStatus** indicates the result status. For details, see **Table 12-3**.

**Table 12-3 Result status list**

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Execution success</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>GA tuning failed to preempt the device.</td>
<td>During GA tuning, the device resources need to be exclusively occupied by Auto Tune. Stop other processes and try GA tuning again.</td>
</tr>
<tr>
<td>2</td>
<td>Failed to register the operator binary file (.o).</td>
<td>Check if the user who performs the tuning has the write permission on the target directory.</td>
</tr>
<tr>
<td>3</td>
<td>Failed to execute the operator binary file on the RTS side.</td>
<td>If an error occurs during operator execution, search for the keywords <code>aic_error</code> and <code>task_exception</code> in the host log file for more information.</td>
</tr>
<tr>
<td>4</td>
<td>Failed to allocate the memory required for the input and output of the operator binary file on the host.</td>
<td>Check if the host has sufficient memory space.</td>
</tr>
<tr>
<td>Other</td>
<td>-</td>
<td>Contact Huawei technical support.</td>
</tr>
</tbody>
</table>
12.5 Appendixes

12.5.1 Operator Lists

The current version of Auto Tune supports only the auto tuning of AI Core operators whose compute logic is implemented using DSL APIs. This section lists the operators that support RL and GA tuning.

Operators Supporting RL Tuning

- Abs
- AbsGrad
- AcosGrad
- Add
- AsinGrad
- AsinhGrad
- AtanGrad
- BiasAddGrad
- BNTrainingUpdate
- Ceil
- ConcatD
- Cos
- Cosh
- Div
- DynamicRNN
- Elu
- EluGrad
- Equal
- Erf
- Erfc
- Exp
- Expm1
- Floor
- Gelu
- GeluGrad
- Inv
- InvGrad
- L2Loss
- L2Normalize
- L2NormalizeGrad
- Log
- LogSoftmaxV2
- LogSoftmaxGrad
- Log1p
- Maximum
- Mod
- Mul
- Neg
- OnesLike
- Pow
- PReluGrad
- Reciprocal
- ReciprocalGrad
- ReduceAllD
- ReduceAnyD
- ReduceMaxD
- ReduceMeanD
- ReduceSumD
- Relu
- ReluGrad
- Relu6
- Relu6Grad
- Rint
- Round
- Rsqrt
- RsqrtGrad
- Selu
- Sigmoid
- SigmoidCrossEntropyWithLogits
- SigmoidGrad
- Sign
- Sinh
- SmoothL1Loss
- SoftmaxV2
- SoftmaxGrad
- Softplus
- Softsign
- SplitD
- Sqrt
- SqrtGrad
- Square
- StridedSliceD
- Sub
- Tanh
- TanhGrad
- SoftmaxCrossEntropyWithLogits
- MatMul
- GNTrainingReduce
- GNTrainingUpdate
- CosineEmbeddingLoss
- INTrainingReduceV2
- INTrainingUpdateV2

**NOTE**
The preceding list summarizes all operators supporting RL tuning, but might vary according to the Ascend AI Processor version. For details, see *Operator Lists*.

**Operators Supporting GA Tuning**
- AvgPool
- Conv2D
- Conv2DBackpropInput
- Conv2DBackpropFilter
- Conv2DCompress
- Conv2DTranspose
- Conv3D
- Conv3DBackpropInput
- Conv3DBackpropFilter
- DepthwiseConv2D
- DepthwiseConv2DBackpropInput
- DepthwiseConv2DBackpropFilter
- Deconvolution
- GEMM
- Pooling

**NOTE**
The preceding list summarizes all operators supporting GA tuning, but might vary according to the Ascend AI Processor version. For details, see *Operator Lists*. 
13 Model Accuracy Analyzer Instructions

13.1 Overview

Huawei supports the migration of the original network for training on the Ascend 910 AI Processor. As a result, the computation result of Huawei proprietary operators may be different from that of third-party operators (for example, from TensorFlow). The Model Accuracy Analyzer is designed to compare the computation results for developers to quickly resolve the operator accuracy issues.

The Model Accuracy Analyzer compares the computation result of the model running on the Ascend 910 AI Processor with that of the original model running on the GPU or CPU to locate the error operators.

Currently, the following vector comparison methods are provided: cosine similarity, maximum absolute error, accumulated relative error, Euclidean relative distance, Kullback-Leibler divergence (KLD), and standard deviation.
NOTE

This document describes how to use Model Accuracy Analyzer with the assumption that the tool is installed by HwHiAiUser in the default installation path /home/HwHiAiUser/Ascend. You can replace them as required. Ensure that HwHiAiUser has the read, or read and write permissions on the paths described in this document.

For an optimal experience, the following hardware configuration is recommended: 8-core CPU at 2.6 GHz with 16 GB memory.

The following data types are supported:

- FLOAT
- FLOAT16
- DT_INT8
- DT_UINT8
- DT_INT16
- DT_UINT16
- DT_INT32
- DT_UINT32
- DT_INT64
- DT_UINT64
- DT_BOOL
- DT_DOUBLE

13.2 Data Preparation

13.2.1 Preparing .npy Data of a TensorFlow Network on GPU

Prerequisites

- Before generating the dump data or .npy files of a trained TensorFlow network, a complete, executable, standard TensorFlow model training project is required.
- Regardless of whether the Estimator or Session.run mode is used, disable all random functions in the script, including but not limited to shuffle operations on datasets, random initialization of parameters, and implicit random initialization of some operators (such as the dense operator). Ensure that all parameters in the script are not initialized randomly.

.npy File Preparation

You can use the TensorFlow debugger (tfdbg) to generate .npy files. The major steps are as follows:

Step 1  Add the debugging configuration option to the TensorFlow training project script.
- If the Estimator mode is used, add the hook of tfdbg as follows:
  a. Add `from tensorflow.python import debug as tf_debug` to import the debug module.
  b. Add the code `raining_hooks=[tf_debug.LocalCLIDebugHook()]` when the EstimatorSpec object instance is generated, that is, when the network structure is constructed.
If the `Session.run` mode is used, set the tfdbg decorator before running as follows:

a. Add `from tensorflow.python import debug as tf_debug` to import the debug module.
b. After the session is initialized, add `sess = tf_debug.LocalCLIDebugWrapperSession(sess, ui_type="readline")`.

**Step 2** Execute the training script. After the training job is stopped, enter `run` in the command line. The training proceeds to the next step.

For more details, see help.

tfdbg> run
**Step 3** Collect .npy files.

After the command is executed and returns the training result of the first step, you can run the `lt` command to query the stored tensors, run the `pt` command to view the tensor content, and save it as a file in NumPy format. The tfdbg dumps only one tensor at a time. To dump all tensors, perform the following steps:

1. Run the `lt > gpu_dump` command to temporarily store all tensor names to the `gpu_dump` file. The command output is as follows:
   ```plaintext
   Wrote output to tensor_name
   ```

2. Exit the tfdbg command line, in the Linux command line, run the following command in the directory where `gpu_dump` is stored to generate commands to run in tfdbg:
   ```plaintext
   timestamp=$[$(date +%s%N)/1000] ; cat gpu_dump | awk '{print "pt",$4,$4}' | awk '{gsub("/", ",", $3);gsub(\"\.", ",", $3);print($1,$2,"-n 0 -w ","$3",""timestamp":""$timestamp"".npy")}'
   ```

3. Copy all generated commands starting with `pt` and paste them to the tfdbg command line. Run the commands to save all .npy files. The files are saved to the directory where the training script is stored.

   By default, .npy files are stored using `numpy.save()`. Slashes (/) and colons (:) are replaced by underscores (_).

   **NOTE**
   
   If the command cannot be pasted on the command-line interface (CLI), run the `mouse off` command in the tfdbg command line to disable the mouse mode before pasting again.

4. Check that names of the generated .npy files comply with the naming rules, as shown in **Figure 13-3**.

   **NOTE**
   
   - An .npy file is named in the format of `{op_name}.{output_index}.\{timestamp\}.npy`, where `op_name` must comply with the A-Za-z0-9_ - regular expression, `timestamp` must comply with the [0-9]{1,255} regular expression, and `output_index` is a digit ranging from 0 to 9.
   - If the name of an .npy file exceeds 255 characters due to a long operator name, comparison of this operator is not supported.
   - The name of some .npy files may not meet the naming requirements due to the tfdbg or operating environment. You can manually rename the files based on the naming rules. If there are a large number of .npy files that do not meet the requirements, generate .npy files again by referring to **13.4.3 How Do I Handle Exceptions in the Generated .npy File Names in Batches?**
13.2.2 Preparing Dump Data and Computational Graphs of a Model on the Ascend AI Processor

Prerequisites

Before dumping data of a trained network after migration, ensure that the model is developed, built, and executed, and the training project is executable.

**NOTE**

- If the training network contains random factors, remove them before dumping.
- Ensure that your code is the same as the code trained on the GPUs in terms of the network structure, operator, optimizer, and parameter initialization policy. Otherwise, the comparison is meaningless.
- Do not perform training and validation at the same time in a training script. That is, do not put training and validation in the same script. Otherwise, two groups of dump data will be generated and you cannot distinguish between them.
- Currently, only AI CPU and AI Core operators can be dumped. Operators such as Huawei Collective Communication Library (HCCL) operators cannot be dumped.

Dump Parameter Configuration

**Step 1** To enable the training script to dump computational graphs, introduce the OS to the package reference area in the training script and set the `DUMP_GE_GRAPH` parameter before building a model. In this way, during the training process, the computational graph file is saved in the directory where the training script is located.

```python
import os
...
def main():
    os.environ['DUMP_GE_GRAPH'] = '2'
```

**Step 2** Modify the script to enable the dump function. Add the lines in bold in the corresponding positions of the script.

- In **Estimator** mode, collect dump data using `dump_config` in `NPURunConfig`. Before `NPURunConfig` is created, instantiate a `DumpConfig` class for dump configuration, including the dump path, iterations to dump, and the dump mode (operator inputs or outputs). For details about each field in the
constructor function of the DumpConfig class, see the Network Porting and Training Guide.

from npu_bridge.estimator.npu.npu_config import DumpConfig

# dump_path: dump path. Create the specified path in advance in the training environment (either in a container or on the host). The running user configured during installation must have the read and write permissions on this path.
# enable_dump: dump enable.
# dump_step: iterations to dump.
# dump_mode: dump mode, selected from input, output, and all

dump_config = DumpConfig(enable_dump=True, dump_path = "/home/HwHiAiUser/output", dump_step="0|5|10", dump_mode="all")

cfg = NPURunConfig(dump_config=dump_config,
                   session_config=session_config)

In Session.run mode, set the dump parameters by setting the session configuration items enable_dump, dump_path, dump_step, and dump_mode.

```python
cfg = tf.ConfigProto()
custom_op = config.graph_options.rewrite_options.custom_optimizers.add()
custom_op.name = "NpuOptimizer"
custom_op.parameter_map["use_off_line"].b = True

custom_op.parameter_map["enable_dump"].b = True

custom_op.parameter_map["dump_path"].s = tf.compat.as_bytes("/home/HwHiAiUser/output")

custom_op.parameter_map["dump_step"].s = tf.compat.as_bytes("0|5|10")

custom_op.parameter_map["dump_mode"].s = tf.compat.as_bytes("all")
```

Table 13-1 Parameter description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable_dump</td>
<td>Data dump enable.</td>
</tr>
<tr>
<td></td>
<td>- <strong>True</strong>: enabled. The dump file path is read from <strong>dump_path</strong>.</td>
</tr>
<tr>
<td></td>
<td>- <strong>False</strong> (default): disabled.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dump_path</td>
<td>Dump path. Required if enable_dump is set to True. The specified path must be created in advance in the environment (either in a container or on the host) where training is performed. The running user configured during installation must have the read and write permissions on this path. The path can be an absolute path or a relative path relative to the path where the command is executed.</td>
</tr>
<tr>
<td></td>
<td>– An absolute path starts with a slash (/), for example, /home/HwHiAiUser/output.</td>
</tr>
<tr>
<td></td>
<td>– A relative path starts with a directory name, for example, output.</td>
</tr>
<tr>
<td>dump_step</td>
<td>Iterations to dump. Defaults to None, indicating that all iterations are dumped. Separate multiple iterations using vertical bars (</td>
</tr>
<tr>
<td>dump_mode</td>
<td>Dump mode. The values are as follows:</td>
</tr>
<tr>
<td></td>
<td>– input: dumps only operator inputs.</td>
</tr>
<tr>
<td></td>
<td>– output (default): dumps only operator outputs.</td>
</tr>
<tr>
<td></td>
<td>– all: dumps both operator inputs and outputs.</td>
</tr>
</tbody>
</table>

Performing Training to Generate Dump Data

**Step 1** Run the training script to generate the dump data file and computational graph file.

- Computational graph file: The file whose name starts with ge is the computational graph file generated when DUMP_GE_GRAPH is set to 2. The file is stored in the directory where the training script is stored.
- Dump data file: The dump data file is generated in the directory specified by dump_path, that is, the {dump_path}/{time}/{deviceid}/{model_name}/ {model_id}/{data_index} directory. For example, if dump_path is set to /home/HwHiAiUser/output, the dump data file is stored in the /home/HwHiAiUser/output/20200808163566/0/ge_default_20200808163719_121/11/0 directory.
<table>
<thead>
<tr>
<th>Path Key</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>dump_path</td>
<td>Dump path set in Step 2. (If a relative path is set, the corresponding absolute path applies.)</td>
<td>--</td>
</tr>
<tr>
<td>time</td>
<td>Dump time.</td>
<td>Format: YYYYMMDDHHMMSS</td>
</tr>
<tr>
<td>deviceid</td>
<td>Device ID.</td>
<td>--</td>
</tr>
<tr>
<td>model_name</td>
<td>Subnetwork name.</td>
<td>If the model_name directory contains more than one folder, dump data in the folder with the same name as the computational graph is used. Periods (.), forward slashes (/), backslashes (), and spaces in model_name are replaced with underscores (_).</td>
</tr>
<tr>
<td>model_id</td>
<td>Subnetwork ID.</td>
<td>--</td>
</tr>
<tr>
<td>data_index</td>
<td>Iterations to dump.</td>
<td>If dump_step is specified, data_index equals to dump_step. If it is not specified, data_index starts at 0 and is incremented by 1 with each dump.</td>
</tr>
<tr>
<td>dump_file</td>
<td>Format: {op_type}.{op_name}. {taskid}.{stream_id}. {timestamp}. If the length of a file name formatted as required exceeds the OS file name length limit (generally 255 characters), the dump file is renamed a string of random digits. For details about the mapping, see the mapping.csv file in the same directory.</td>
<td>Periods (.), forward slashes (/), backslashes (), and spaces in op_type or op_name are replaced with underscores (_).</td>
</tr>
</tbody>
</table>
**NOTE**

- Dump data is generated in each iteration. A large training dataset generates a large volume of dump data (about dozens of GB or even more). You are advised to control the number of iterations to one.
- In the multi-device training scenario where more than one Ascend AI Processor is used, since the processes are not started at the same time as defined in the training script, multiple timestamp directories are generated when data is dumped.
- When the command is executed in a Docker, the generated data is stored in the Docker.

**Step 2** Select a computational graph file.

**NOTE**

There are a large number of dump graph files whose names start with `ge`, and multiple folders may exist at the `model_name` layer in the dump data file. In fact, you only need to find the computational graph file and the folder whose `model_name` is the name of the computational graph. You can use either of the following methods to quickly find the required file:

- **Method 1**: Search for the keyword `Iterator` in all dump files whose names end with `_Build.txt`. Record the name of the computational graph file, which will be used for accuracy comparison.

  ```bash
grep Iterator *_Build.txt
```

  As shown in the preceding figure, the `ge_proto_00292_Build.txt` file is the desired computational graph file.

- **Method 2**: Save the TensorFlow model as a PB file, view the model, select the name of a computing operator as the keyword, and find the computational graph file that contains the keyword. The value of the `name` field in the computational graph is used as the name of the computational graph.

**Step 3** Select the dump data file.

1. Open the computational graph file found in **Step 2** and record the value of the `name` field in the first graph. In the following example, record the value `ge_default_20201209083353_71`.

   ```
   graph {
       name: "ge_default_20201209083353_71"
   
       op {
           name: "atomic_addr_clean0_71"
           type: "AtomicAddrClean"
           attr { key: "_fe_imply_type" value { i: 6 } }
       }
   }
   ```

2. Go to the directory for storing the dump file named after the timestamp. The following folders exist in the directory:
3. Find the folder whose name is the recorded value, for example, `ge_default_20201209083353_71`. The files in the folder are the required dump data files.

### 13.3 Vector Comparison

Vector comparison by model and by single operator are supported. You can select a comparison mode as required.

Note the following restrictions on vector comparison:

The dump data generated by running on Ascend AI Processor and the dump data generated by running on GPU or CPU should be obtained from the counterpart models.

#### 13.3.1 Network-wide Comparison

**Command Syntax**

The command for vector comparison is structured as follows:

```python
python3.7.5 msaccucmp.py compare [-m my_dump_path] [-g golden_dump_path] [-f fusion_rule_file] [-out output] [-c custom_script_path] [-v version]
```

*Table 13-3* describes the command-line options.
Table 13-3 Command-line options for network-wide comparison

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-m --my_dump_path</td>
<td>Directory of the dump data of the model running on the Ascend AI Processor, that is, the directory of the dump data queried in Step 3.</td>
<td>Yes</td>
</tr>
<tr>
<td>-g --golden_dump_path</td>
<td>Directory of the .npy file of the TensorFlow model running on the GPUs or CPUs.</td>
<td>Yes</td>
</tr>
<tr>
<td>-f --fusion_rule_file</td>
<td>Network-wide information file (.json file generated from the .txt graph file using ATC), that is, the computational graph queried in Step 2.</td>
<td>Yes</td>
</tr>
<tr>
<td>-out --output</td>
<td>Path of the comparison result. Defaults to the current path.</td>
<td>No</td>
</tr>
<tr>
<td>-c --custom_script_path</td>
<td>User-defined path to store the .py file for format conversion, which should be the parent directory of the format_convert directory. For details about the .py file requirements, see Preparing a Customized .py File for Format Conversion.</td>
<td>No</td>
</tr>
<tr>
<td>-v --version</td>
<td>Dump file type. 1: data file after Protobuf serialization; 2 (default): data file in a custom format.</td>
<td>No</td>
</tr>
</tbody>
</table>

Comparison Procedure

To conduct vector comparison, perform the following steps.

NOTE

- The .json file and directory names in this section are only examples. Replace them with the actual ones. Ensure that the HwHiAiUser user has the read and write permissions on the result path specified by --out.
- If “MemoryError” is displayed during comparison, memory overflow occurs due to overloaded data. Split dump files on the NPU into different directories and compare the files one by one.

Step 1 Log in to the OS as the HwHiAiUser user.
Step 2 Run the `export` command to set the environment variable and generate a .json file.

Set the following environment variable:

```bash
export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/atc/lib64:${LD_LIBRARY_PATH}
```

Convert the graph file into a .json file:

```bash
/home/HwHiAiUser/Ascend/ascend-toolkit/latest/atc/bin/atc --mode=5 --om=ge_proto_00005_Build.txt --json=ge_proto_00005_Build.txt.json
```

**NOTE**

The `ge_proto_00005_Build.txt` file name in the preceding command line is used as an example. You can replace it as required.

After the training script is executed, you might find that more than one GE graph file is generated to the training script directory. To select the right computational graph file, save the TensorFlow model as a .pb file and view the .pb model. Choose the name of a random compute operator as the search keyword, and search for the keyword in the generated graph files. The graph that gives a match is the desired computational graph file, whose name is indicated by the `name` field under `graph`.

Step 3 Go to the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare` directory.

Step 4 Run the vector comparison command as follows:

```bash
python3.7 msaccucmp.py compare -m /home/HwHiAiUser/MyApp_mind/resnet50 -g /home/HwHiAiUser/Standard_tf/resnet50 -f /home/HwHiAiUser/data/ge_proto_00005_Build.txt.json -out /home/HwHiAiUser/result
```

The vector comparison result is saved to the `result_*.csv` file, as shown in Figure 13-4.

**Figure 13-4 Model comparison result**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftOp</td>
<td>Name of the dumped operator running on the Ascend AI Processor.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RightOp</td>
<td>Name of the operator (running on the GPUs or CPUs) that generates the dump data or .npy file.</td>
</tr>
<tr>
<td>TensorIndex</td>
<td>Input ID and output ID of the dumped operator running on the Ascend AI Processor.</td>
</tr>
<tr>
<td>CosineSimilarity</td>
<td>Result of the cosine similarity comparison. The value range is [-1, +1]. A value closer to 1 indicates higher similarity.</td>
</tr>
<tr>
<td>MaxAbsoluteError</td>
<td>Result of the maximum absolute error comparison. The value ranges from 0 to infinity. A value closer to 0 indicates higher similarity.</td>
</tr>
<tr>
<td>AccumulatedRelativeError</td>
<td>Result of the accumulated relative error comparison. The value ranges from 0 to infinity. A value closer to 0 indicates higher similarity.</td>
</tr>
<tr>
<td>RelativeEuclideanDistance</td>
<td>Result of the Euclidean relative distance comparison. The value ranges from 0 to infinity. A value closer to 0 indicates higher similarity.</td>
</tr>
<tr>
<td>KullbackLeiblerDivergence</td>
<td>Result of the KLD comparison. The value ranges from 0 to infinity. The smaller the KLD is, the closer the approximate distribution is to the true distribution.</td>
</tr>
<tr>
<td>StandardDeviation</td>
<td>Result of the standard deviation comparison. The value ranges from 0 to infinity. The smaller the standard deviation is, the smaller the dispersion is, and the closer the value is to the average value. The mean value and standard deviation of the dump data are displayed in the format of (mean value;standard deviation). The first set of data is the result of the model running on the Ascend AI Processor, and the second set is the result of the model running on the GPUs or CPUs.</td>
</tr>
<tr>
<td>CompareFailReason</td>
<td>Comparison failure cause.</td>
</tr>
</tbody>
</table>

Notes:
- An asterisk (*) indicates a newly added operator with no third-party counterpart. NaN indicates there is no comparison result.
- If the results of cosine similarity and KLD are NaN, and the results of other algorithms exist, at least one piece of data on the left or the right is 0. If the result of KLD is inf, one piece of data on the right is 0.

13.3.2 Single-Operator Comparison

Command Syntax

The command for vector comparison is structured as follows:
Table 13-5 describes the command-line options.

**Table 13-5 Command-line options for single-operator comparison**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-m --my_dump_path</td>
<td>Directory of the dump data of the model running on the Ascend AI Processor, that is, the directory of the dump data queried in Step 3.</td>
<td>Yes</td>
</tr>
<tr>
<td>-g --golden_dump_path</td>
<td>Directory of the npy file of the TensorFlow model running on the GPUs or CPUs.</td>
<td>Yes</td>
</tr>
<tr>
<td>-f --fusion_rule_file</td>
<td>Network-wide information file (.json file generated from the .txt graph file using ATC), that is, the computational graph queried in Step 2.</td>
<td>Yes</td>
</tr>
<tr>
<td>-out --output</td>
<td>Path of the comparison result. Defaults to the current path.</td>
<td>No</td>
</tr>
<tr>
<td>-op --op_name</td>
<td>Name of the single-operator.</td>
<td>No</td>
</tr>
</tbody>
</table>
| -o --output_tensor | Index of the output to compare. Mutually exclusive with -i. Valid only when -op is configured. 
If neither -o nor -i is included, the output indexed 0 is compared. | No       |
| -i --input_tensor  | Index of the input to compare. Mutually exclusive with -o. Valid only when -op is configured. | No       |
| -c --custom_script_path | User-defined path to store the .py file for format conversion, which should be the parent directory of the format_convert directory. For details about the .py file requirements, see Preparing a Customized .py File for Format Conversion. | No       |
| -v --version       | Dump file type. 1: data file after Protobuf serialization; 2 (default): data file in a custom format. | No       |

**Comparison Procedure**

To conduct vector comparison, perform the following steps.
NOTE

The .json file and directory names in this section are only examples. Replace them with the actual ones. Ensure that the HwHiAiUser user has the read and write permissions on the result path specified by --out.

Single-operator comparison between two groups of dump data generated through the same training job by running on the Ascend AI Processor is not supported.

Step 1 Log in to the OS as the HwHiAiUser user.

Step 2 Run the export command to set the environment variable and generate a .json file.

Set the following environment variable:

```bash
export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/atc/lib64:${LD_LIBRARY_PATH}
```

Convert the model file into a .json file:

```bash
/home/HwHiAiUser/Ascend/ascend-toolkit/latest/atc/bin/atc --mode=5 --om=ge_proto_00005_Build.txt --json=ge_proto_00005_Build.txt.json
```

NOTE

The ge_proto_00005_Build.txt file name in the preceding command line is used as an example. You can replace it as required.

After the training script is executed, you might find that more than one GE graph file is generated to the training script directory. To select the right computational graph file, save the TensorFlow model as a .pb file and view the .pb model. Choose the name of a random compute operator as the search keyword, and search for the keyword in the generated graph files. The graph that gives a match is the desired computational graph file, whose name is indicated by the name field under graph.

Step 3 Go to the /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare directory.

Step 4 Run the vector comparison command as follows:

```bash
python3.7 msaccucmp.py compare -m /home/HwHiAiUser/MyApp_mind/resnet50 -g /home/HwHiAiUser/Standard_tf/resnet50 -f /home/HwHiAiUser/data/ge_proto_00005_Build.txt.json -out /home/HwHiAiUser/result -op gradients/AddN_63 -i 0
```

Figure 13-5 and Figure 13-6 show the content of a vector comparison result file.

Figure 13-5 Single-operator comparison result (summary)

<table>
<thead>
<tr>
<th>TotalCount</th>
<th>100352</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftOp:pool15</td>
<td></td>
</tr>
<tr>
<td>RightOp:pool15</td>
<td></td>
</tr>
<tr>
<td>Format:N,C,H,W</td>
<td></td>
</tr>
<tr>
<td>MinAbsoluteError</td>
<td>0.00000</td>
</tr>
<tr>
<td>MaxAbsoluteError</td>
<td>0.077095</td>
</tr>
<tr>
<td>MinRelativeError</td>
<td>0.00000</td>
</tr>
<tr>
<td>MaxRelativeError</td>
<td>15.401893</td>
</tr>
</tbody>
</table>

The single-operator comparison result summary is stored in {op_name}_input_{index}_summary.txt or
The parameters are described as follows:

Table 13-6 Single-operator comparison result summary parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalCount</td>
<td>Number of data records in the dump data of the operator.</td>
</tr>
<tr>
<td>LeftOp</td>
<td>Name of the dumped operator running on the Ascend AI Processor.</td>
</tr>
<tr>
<td>RightOp</td>
<td>Name of the operator (running on the GPUs or CPUs) that generates the dump data or .npy file.</td>
</tr>
<tr>
<td>Format</td>
<td>Data format.</td>
</tr>
<tr>
<td>MinAbsoluteErr</td>
<td>Minimum absolute error.</td>
</tr>
<tr>
<td>MaxAbsoluteErr</td>
<td>Maximum absolute error.</td>
</tr>
<tr>
<td>MinRelativeErr</td>
<td>Minimum relative error.</td>
</tr>
<tr>
<td>MaxRelativeErr</td>
<td>Maximum relative error.</td>
</tr>
</tbody>
</table>

Figure 13-6 Single-operator comparison result (details)

<table>
<thead>
<tr>
<th>Index</th>
<th>N</th>
<th>C</th>
<th>H</th>
<th>Left</th>
<th>Right</th>
<th>AbsoluteError</th>
<th>RelativeError</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0.784180</td>
<td>0.784180</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.054688</td>
<td>0.000000</td>
<td>1.054688</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4.000000</td>
<td>0.000000</td>
<td>4.000000</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3.000000</td>
<td>2.000000</td>
<td>1.000000</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4.000000</td>
<td>3.563013</td>
<td>1.273332</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>5.000000</td>
<td>0.000000</td>
<td>5.000000</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>6.000000</td>
<td>3.359385</td>
<td>2.641189</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>7.000000</td>
<td>5.650749</td>
<td>3.353184</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>8.000000</td>
<td>0.000000</td>
<td>8.000000</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>9.000000</td>
<td>0.000000</td>
<td>9.000000</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>10.000000</td>
<td>2.242188</td>
<td>7.757812</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>11.000000</td>
<td>3.340527</td>
<td>1.260473</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>12.000000</td>
<td>0.000000</td>
<td>12.000000</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>13.000000</td>
<td>3.476562</td>
<td>4.422243</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>14</td>
<td>0</td>
<td>2</td>
<td>14.000000</td>
<td>0.000000</td>
<td>14.000000</td>
<td>-</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>0</td>
<td>2</td>
<td>15.000000</td>
<td>0.000000</td>
<td>15.000000</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>16</td>
<td>0</td>
<td>2</td>
<td>16.000000</td>
<td>0.000000</td>
<td>16.000000</td>
<td>-</td>
</tr>
<tr>
<td>19</td>
<td>17</td>
<td>0</td>
<td>2</td>
<td>17.000000</td>
<td>3.228516</td>
<td>3.926752</td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>18</td>
<td>0</td>
<td>2</td>
<td>18.000000</td>
<td>5.765625</td>
<td>5.42937</td>
<td>-</td>
</tr>
<tr>
<td>21</td>
<td>19</td>
<td>0</td>
<td>2</td>
<td>19.000000</td>
<td>3.778811</td>
<td>3.260678</td>
<td>-</td>
</tr>
<tr>
<td>22</td>
<td>20</td>
<td>0</td>
<td>2</td>
<td>20.000000</td>
<td>0.000000</td>
<td>20.000000</td>
<td>-</td>
</tr>
</tbody>
</table>
The detailed comparison result of the single-operator is stored in `{op_name}_input_{index}_{file_index}.csv` or `{op_name}_output_{index}_{file_index}.csv`. Each file records a maximum of one million data records. The parameters in Figure 13-6 are described as follows.

### Table 13-7 Single-operator comparison result detail parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N C H W</td>
<td>Data coordinates.</td>
</tr>
<tr>
<td>Left</td>
<td>Dump value of the operator running on the Ascend AI Processor.</td>
</tr>
<tr>
<td>Right</td>
<td>Dump value of the operator running on the GPUs or CPUs.</td>
</tr>
<tr>
<td>RelativeError</td>
<td>Relative error. The value is obtained by dividing the AbsoluteError value by the dump value of the operator in the Right column. If the dump value of the operator in the Right column is 0, a hyphen (-) is displayed.</td>
</tr>
<tr>
<td>AbsoluteError</td>
<td>Absolute error. The value is the difference between the dump value of the operator in the Left column and the dump value of the operator in the Right column.</td>
</tr>
</tbody>
</table>

---End

### 13.4 Appendixes

#### 13.4.1 How Do I Convert the Format of a Dump File?

**Converting the Format of a Dump File**

In the current version, dump files generated by running on the Ascend AI Processor can be converted into NumPy format.

To perform the conversion, execute the `msaccucmp.py` script stored in the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare` path. The command syntax is as follows:

```bash
python3.7.5 msaccucmp.py convert [-d dump_file] [-t type] [-c custom_script_path] [-v version] [-i input_tensor] [-o output_tensor] [-f format -s shape] [-out output]
```

**Table 13-8** describes the command-line options.
### Table 13-8 Command-line options for format conversion

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d --dump_file</td>
<td>Dump file (including the path) of a model running on the the Ascend AI Processor.</td>
<td>Yes</td>
</tr>
<tr>
<td>-out --output</td>
<td>Directory of the converted data. Defaults to the current path.</td>
<td>No</td>
</tr>
</tbody>
</table>
| -f --format        | - If the command contains -f, data is converted to a specified format. If the dump file contains the original_shape field, the data is tiled based on original_shape.  
- If the command does not contain -f, the dump file is parsed.                                                                                     | No       |
| -s --shape         | Shape to be set for FRACTAL_NZ conversion. The shape format is ([0-9]+,)+[0-9]+, where each number must be greater than 0. Valid only when -f is configured.                                                        | No       |
| -o --output_tensor | Index of the output to convert. Mutually exclusive with -i. Valid only when -f is configured.                                                                                                           | No       |
| -i --input_tensor  | Index of the input to convert. Mutually exclusive with -o. Valid only when -f is configured.                                                                                                                  | No       |
| -c --custom_script_path | User-defined path to store the .py file for format conversion, which should be the parent directory of the format_convert directory. For details about the .py file requirements, see Preparing a Customized .py File for Format Conversion. Valid only when -f is configured. | No       |
| -v --version       | Dump file type. 1: data file after Protobuf serialization; 2 (default): data file in the custom format.                                                                                                     | No       |
| -t --type          | Type of the dump file. **npy** (default): dump file in NumPy format; **bin**: dump file in binary format.                                                                                                | No       |
The result is saved in the format of `original_file_name.output.{index}.{shape}.npy` or `original_file_name.input.{index}.{shape}.npy`, where `shape` is formatted as `1x3x224x224`.

Currently, the following built-in format conversion types are supported:

- FRACTAL_NZ to NCHW
- FRACTAL_NZ to NHWC
- FRACTAL_NZ to ND
- HWCN to FRACTAL_Z
- HWCN to NCHW
- HWCN to NHWC
- NC1HWC0 to HWCN
- NC1HWC0 to NCHW
- NC1HWC0 to NHWC
- NCHW to FRACTAL_Z
- NCHW to NHWC
- NHWC to FRACTAL_Z
- NHWC to HWCN
- NHWC to NHWC

Preparing a Customized .py File for Format Conversion

Prepare the file as follows:

- The name of the .py file is in `convert_{format_from}_to_{format_to}.py` format. The supported formats for `format_from` and `format_to` are as follows:
  - NCHW
  - NHWC
  - ND
  - NC1HWC0
  - FRACTAL_Z
  - NC1C0HWPAD
  - NHWC1C0
  - FSR_NCHW
  - FRACTAL_DECONV
  - C1HWNC0
  - FRACTAL_DECONV_TRANSPOSE
  - FRACTAL_DECONV_SP_STRIDE_TRANS
  - NC1HWC0_C04
  - FRACTAL_Z_C04
  - CHWN
  - DECONV_SP_STRIDE8_TRANS
  - NC1KHWHWC0
  - BN_WEIGHT
The content of the .py file is as follows:

def convert(shape_from, shape_to, array):
    return numpy_array

The parameters are described as follows:

- **shape_from**: shape of the one-dimensional array before conversion
- **shape_to**: (optional) shape of the one-dimensional array after conversion
- **array**: one-dimensional source data
- **return**: NumPy array after conversion

The directory of the .py file must meet the following requirement:

- The .py file must be stored in the `format_convert` directory. If the directory does not exist, create one.

### 13.4.2 How Do I View a Dump File?

Dump files cannot be viewed with a text tool. Therefore, you need to convert your dump file into a NumPy file and save the NumPy file as a text file using `numpy.savetxt`.

To perform the conversion, execute the `msaccucmp.py` script stored in the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare` path. The command syntax is as follows:

```
python3.7.5 msaccucmp.py convert -d dump_file [-out output] [-v version] [-t type]
```

**Table 13-9** describes the command-line options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d/--dump_file</td>
<td>Dump file (including the path) of a model running on the Ascend AI Processor.</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Procedure

**Step 1** Log in to the development environment as the installation user.

**Step 2** Go to the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare` directory.

**Step 3** Run the `msaccucmp.py` script to convert the dump file into a NumPy file. The following is an example:

```python
python3.7.5 msaccucmp.py convert -d /home/HwHiAiUser/dump -out /home/HwHiAiUser/dumptonumpy -v 2
```

**NOTE**
- For details about the parameters of the `msaccucmp.py` script, see 13.4.1 How Do I Convert the Format of a Dump File?
- The `-d` option enables the conversion of a single dump file or all dump files in a path.

**Step 4** Use Python to save the NumPy data into a text file. The following is an example:

```python
$ python3.7.5
Python 3.7.5 (default, Mar 5 2020, 16:07:54) [GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> a = np.load("/home/HwHiAiUser/dumptonumpy/Pooling.pool1.1147.1589195081588018.output.0.npy")
>>> b = a.flatten()
>>> np.savetxt("/home/HwHiAiUser/dumptonumpy/Pooling.pool1.1147.1589195081588018.output.0.txt", b)
```

The dimension information and `Dtype` no longer exist in the `.txt` file. For details, visit the NumPy website.

### 13.4.3 How Do I Handle Exceptions in the Generated `.npy` File Names in Batches?

When generating dump data of a TensorFlow model, the names of some `.npy` files may be truncated due to the tfdbg or operating environment. As a result, the `.npy`
file names may not meet the naming requirements, and dump file conversion may fail. In this case, perform the following steps to re-generate the .npy files:

**NOTE**

- The script names and paths in this document are used as examples. Replace them as required.
- After batch processing, if the dump file of an operator exists but the comparison result is NaN, check whether the `{op_name}` field of the dump file name of the operator is the same as the TensorFlow operator name. If not, manually change `{op_name}` to the TensorFlow operator name. If a slash (/) exists, replace it with an underscore (_).

**Step 1** Execute the TensorFlow project.

In the interactive debugger command line, enter `run` to run the script.

**Step 2** Run `lt > tensor_name` to temporarily store all tensor names to a file.

**Step 3** Create an executable script, for example, `pt_cmd.sh`, to obtain the tensor_index corresponding to `tensor_name` in the `tensor_name` file.

The script content is as follows.

```bash
#!/bin/bash
timestamp=$(date +%s%N)/1000
index=1
while read -r line
do
tensor_index=`echo $line | awk '{print $4}'`
echo "pt $tensor_index -n 0 -w "$((index++))"$timestamp".npy" >> $2
done < $1
```

Grant the execute permission on the `pt_cmd.sh` script and execute the script.

```bash
chmod +x pt_cmd.sh
bash pt_cmd.sh tensor_name tensor_name.txt
```

**Step 4** Go back to the tfdbg command line, run the script, and paste and execute the content in the `tensor_name.txt` file generated in the previous step to save all .npy files.

**Step 5** Move the generated .npy files to a new folder, for example, `npy_dir`.

**Step 6** Create an executable script, for example, `index_to_tensorname.sh`, and run the script to change the .npy file names in batches.

The script content is as follows.

```bash
#!/bin/bash
timestamp=$(date +%s%N)/1000
index=1
while read -r line
do
tensor_index=`echo $line | awk '{print $2}'`
real_file=`echo $line | awk '{print $6}'`
changed1_tensor_index=${tensor_index////_}
changed2_tensor_index=${changed1_tensor_index//:/.}
echo $2/$real_file $2/$changed2_tensor_index"$timestamp".npy"
if [ -r $2/$real_file ]
then
  mv $2/$real_file $2/$changed2_tensor_index"$timestamp".npy"
fi
done < $1
```
Grant the execute permission on the `index_to_tensorname.sh` script and execute the script.

```
chmod +x index_to_tensorname.sh
bash index_to_tensorname.sh tensor_name.txt npy_dir
```

----End

### 13.4.4 How Do I Migrate a Dump File from Windows to Linux?

**Step 1** Prepare the conversion script `windows_to_linux.py` on Linux as follows.

```python
import argparse
import os
import sys

class WindowsToLinux:
    """
The class for format windows to linux
    """
    def __init__(self):
        parse = argparse.ArgumentParser()
        parse.add_argument("-i", dest="windows_dump_path", default="", help="<Required> the dump file path", required=True)
        parse.add_argument("-o", dest="output_path", default="", help="<Required> the output path", required=True)
        args, _ = parse.parse_known_args(sys.argv[1:])
        self.windows_dump_path = os.path.realpath(args.windows_dump_path)
        self.output_path = os.path.realpath(args.output_path)

    def windows_to_linux(self, dump_path):
        try:
            with open(dump_path, "rb") as dump_file:
                content = dump_file.read()
                new_content = content.replace(b"\r\n", b"\n")
                output_file_path = os.path.join(self.output_path, os.path.basename(dump_path))
                with open(output_file_path, "wb") as new_dump_file:
                    new_dump_file.write(new_content)
                print('Info: convert doc to linux for "%s" successfully.' % dump_path)
        except (OSError, IOError, MemoryError, SystemError) as error:
            print('Error: convert windows dump file "%s" to linux dump file failed. %s' % (dump_path, error))

    def convert(self):
        try:
            if not os.path.exists(self.windows_dump_path) and \
                not os.access(self.windows_dump_path, os.R_OK):
                print('Error: the path "%s" does not exist or can not readable.')
                sys.exit()
            if not os.path.exists(self.output_path):
                os.makedirs(self.output_path)
            if not os.path.exists(self.output_path) and \
                not os.access(self.output_path, os.W_OK):
                print('Error: the path "%s" does not exist or can not writable.')
                sys.exit()
            if os.path.isfile(self.windows_dump_path):
                self.windows_to_linux(self.windows_dump_path)
            else:
                for file_name in os.listdir(self.windows_dump_path):
                    self.windows_to_linux(os.path.join(self.windows_dump_path, file_name))
        except (OSError, IOError, MemoryError, SystemError) as error:
            print('Error: convert windows dump file to linux dump file failed. %s' % error)
            sys.exit()
```

if __name__ == "__main__":
    main = WindowsToLinux()
    main.convert()

Step 2  Copy the dump file of the Windows OS to any directory of the Linux OS.

Step 3  Run the conversion script on Linux.
# Replace windows_dump_path with that specified in Step 2.
# Replace linux_dump_path with that for storing the generated dump file.
python3.7.5 windows_to_linux.py -i {windows_dump_path} -o {linux_dump_path}

----End

13.4.5 compare_vector.py

13.4.5.1 Instructions

Accuracy comparison by model and by single-operator are supported. You can select a comparison mode as required.

Note the following restrictions:

The dump data generated by running on Ascend AI Processor and the dump data generated by running on GPU or CPU should be obtained from the counterpart models.

\[\text{NOTE}\]

13.2 Data Preparation and 13.3 Vector Comparison show new vector comparison modes. For details about former comparison modes, see 13.4.5 compare_vector.py.

Former comparison modes will not be updated. New comparison modes are recommended.

Former comparison modes support only dump files after Protobuf serialization. Dump files in custom formats are not supported.

13.4.5.2 Network-wide Comparison

Command Syntax

The command for vector comparison is structured as follows:

```
python3.7 compare_vector.py -l LEFT_DUMP_PATH -r RIGHT_DUMP_PATH -f FUSION_JSON_FILE_PATH -o OUTPUT_PATH [-custom CUSTOM_PATH]
```

- **-l**: path of the dump files of the model running on the Ascend AI Processor. Ensure that names of the dump files in this path are in `op_type.op_name.taskid.timestamp` format, where `op_type` and `op_name` must comply with the A-Za-z0-9_- regular expression rule, `timestamp` is of 16 bits, and `taskid` is a digit in the range of 0-9.

- **-r**: path of the .npy or dump data of the TensorFlow model running on the GPUs or CPUs. Names of files in the directory must meet the following requirements:
  - Names of the dump files are in `op_name.output_index.timestamp.pb` format, where `op_name` must comply with the A-Za-z0-9_- regular expression rule, `timestamp` is of 16 bits, and `output_index` is a digit in the range 0-9.
- Names of the .npy files are in op_name.output_index.timestamp.npy format, where op_name must comply with the A-Za-z0-9_- regular expression rule, timestamp is of 16 bits, and output_index is a digit in the range 0–9.

- -f: network-wide information file (.json file generated from the .txt graph file using ATC).
- -o: path and file name of the comparison result.
- -custom: (optional) customized path to store the .py file for format conversion, which should be the upper-level directory of the format_convert directory. For details about the .py file requirements, see Preparing a Customized .py File for Format Conversion.

Comparison Procedure

To conduct vector comparison, perform the following steps:

NOTE
Replace the .json file and directory names in this example with the actual ones. The result path must be created in advance and the HwHiAiUser user must have the read and write permissions on the path.

Step 1 Log in to the OS as the HwHiAiUser user.

Step 2 Run the export command to set the environment variable and generate a .json file.

Set the following environment variable:

```
export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/atc/lib64:${LD_LIBRARY_PATH}
```

Convert the graph file into a .json file:

```
/home/HwHiAiUser/Ascend/ascend-toolkit/latest/atc/bin/atc --mode=5 --om=ge_proto_00005_Build.txt --json=ge_proto_00005_Build.txt.json
```

NOTE
The ge_proto_00005_Build.txt file name in the preceding command line is used as an example. You can replace it as required.

Assume that the GE graphs are named ge_proto_*****_Build.txt. The GE graph with the IteratorV2, Iterator, or GetNext in the name field is the computational graph.

Step 3 Go to the /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare directory.

Step 4 Run the vector comparison command as follows:

```
python3.7 compare_vector.py -l /home/HwHiAiUser/MyApp_mind/resnet50 -r /home/HwHiAiUser/Standard_tf/resnet50 -f /home/HwHiAiUser/data/ge_proto_00005_Build.txt.json -o /home/HwHiAiUser/result/result.txt
```

NOTE
To save the file in CSV format, modify the command into -o /home/HwHiAiUser/result/result.csv -csv.
**Step 5** The vector comparison result is saved to the `result.txt` file, as shown in **Figure 13-7**.

**Figure 13-7** Model comparison result

<table>
<thead>
<tr>
<th>Index</th>
<th>LeftOp</th>
<th>RightOp</th>
<th>TensorIndex</th>
<th>CosineSimilarity</th>
<th>MaxAbsoluteError</th>
<th>AccumulatedRelativeError</th>
<th>RelativeEuclideanDistance</th>
<th>KullbackLeiblerDivergence</th>
<th>StandardDeviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>569</td>
<td>Scale</td>
<td>Scale</td>
<td>TensorIndex3</td>
<td>0.299507</td>
<td>0.963606</td>
<td>1.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>570</td>
<td>Scale</td>
<td>Scale</td>
<td>TensorIndex3</td>
<td>0.299507</td>
<td>0.963606</td>
<td>1.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

The parameters are described as follows:

- **LeftOp**: name of the dumped operator running on the Ascend AI Processor.
- **RightOp**: name of the operator (running on the GPUs or CPUs) that generates the dump data or `.npy` file.
- **TensorIndex**: input ID and output ID of the dumped operator running on the Ascend AI Processor.
- **CosineSimilarity**: result of the cosine similarity comparison. The value range is \([-1, +1]\). A value closer to 1 indicates higher similarity. Otherwise, it indicates greater difference.
- **MaxAbsoluteError**: result of the maximum absolute error comparison. A value closer to 0 indicates higher similarity. Otherwise, it indicates greater difference.
- **AccumulatedRelativeError**: result of the accumulated relative error comparison. A value closer to 0 indicates higher similarity. Otherwise, it indicates greater difference.
- **RelativeEuclideanDistance**: result of the Euclidean relative distance comparison. A value closer to 0 indicates higher similarity. Otherwise, it indicates greater difference.
- **KullbackLeiblerDivergence**: result of the KLD comparison. The value ranges from 0 to infinity. The smaller the KLD is, the closer the approximate distribution is to the true distribution.
- **StandardDeviation**: result of the standard deviation comparison. The value ranges from 0 to infinity. The smaller the standard deviation is, the smaller the dispersion is, and the closer the value is to the average value. The mean value and standard deviation of the dump data are displayed in the format of \((\text{mean value};\text{standard deviation})\). The first set of data is the result of the model running on the Ascend AI Processor, and the second set is the result of the model running on the GPUs or CPUs.
- An asterisk (*) indicates a newly added operator with no third-party counterpart. **NaN** indicates there is no comparison result.
- If the results of cosine similarity and KLD are **NaN**, and the results of other algorithms exist, at least one piece of data on the left or the right is 0. If the result of KLD is **inf**, one piece of data on the right is 0.

----End
13.4.5.3 Single-Operator Comparison

Command Syntax

Before running the comparison command, grant the HwHiAiUser user with the read and write permissions on the directory that stores the comparison results.

The command for vector comparison is structured as follows:

```python
python3.7 compare_vector.py -l LEFT_DUMP_PATH -r RIGHT_DUMP_PATH -f FUSION_JSON_FILE_PATH -o OUTPUT_PATH -d OP_NAME [-t DETAIL_TYPE] [-i OUTPUT_INDEX] [-custom CUSTOM_PATH]
```

The command-line options are described as follows:

- `-l`: path of the dump files of the model running on the Ascend AI Processor. Ensure that names of the dump files in this path are in `op_type.op_name.taskid.timestamp` format, where `op_type` and `op_name` must comply with the `A-Za-z0-9_` regular expression rule, `timestamp` is of 16 bits, and `taskid` is a digit in the range of 0–9.
- `-r`: path of the .npy or dump data of the TensorFlow model running on the GPUs or CPUs. Names of files in the directory must meet the following requirements:
  - Names of the dump files are in `op_name.output_index.timestamp.pb` format, where `op_name` must comply with the `A-Za-z0-9_` regular expression rule, `timestamp` is of 16 bits, and `output_index` is a digit in the range 0–9.
  - Names of the .npy files are in `op_name.output_index.timestamp.npy` format, where `op_name` must comply with the `A-Za-z0-9_` regular expression rule, `timestamp` is of 16 bits, and `output_index` is a digit in the range 0–9.
- `-f`: network-wide information file (.json file generated from the .txt graph file using ATC).
- `-o`: path of the comparison result.
- `-d`: name of the single-operator layer to be compared.
- `-t`: (optional) indicates whether the dump file is the input or output (default).
- `-i`: (optional) input or output index of the operator.
- `-custom`: (optional) customized path to store the .py file for format conversion, which should be the upper-level directory of the `format_convert` directory. For details about the .py file requirements, see Preparing a Customized .py File for Format Conversion.

Comparison Procedure

To conduct vector comparison, perform the following steps:

1. Log in to the OS as the HwHiAiUser user.

   **NOTE**

   Replace the .json file and directory names in this example with the actual ones. The result path must be created in advance and the user who runs the command must have the read and write permissions on the path.
Step 2 Run the `export` command to set the environment variable and generate a .json file.

Set the following environment variable:

```
export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/atc/lib64:${LD_LIBRARY_PATH}
```

Convert the model file into a .json file:

```
/home/HwHiAiUser/Ascend/ascend-toolkit/latest/atc/bin/atc --mode=5 --om=ge_proto_00005_Build.txt --json=ge_proto_00005_Build.txt.json
```

□ NOTE

The `ge_proto_00005_Build.txt` file name in the preceding command line is used as an example. You can replace it as required.

Assume that the GE graphs are named `ge_proto_*****_Build.txt`. The GE graph with the `IteratorV2`, `Iterator`, or `GetNext` in the name field is the computational graph.

Step 3 Go to the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare` directory.

Step 4 Run the vector comparison command as follows:

```
python3.7 compare_vector.py -l /home/HwHiAiUser/MyApp_mind/resnet50 -r /home/HwHiAiUser/Standard_tf/resnet50 -f /home/HwHiAiUser/data/ge_proto_00005_Build.txt.json -o /home/HwHiAiUser/result -d gradients/AddN_63 -i 0
```

Step 5 **Figure 13-8** and **Figure 13-9** show the content of a vector comparison result file.

**Figure 13-8** Single-operator comparison result (summary)

```
TotalCount: 100352
LeftOp: pool15
RightOp: pool15
Format: N, C, H, W
MinAbsoluteError: 0.000000
MaxAbsoluteError: 0.077095
MinRelativeError: 0.000000
MaxRelativeError: 15.401893
```

The single-operator comparison result summary is stored in `{op_name}_input_{index}_summary.txt` or `{op_name}_output_{index}_summary.txt`. The parameters are described as follows:

- **TotalCount**: number of data records in the dump data of the operator
- **LeftOp**: name of the dumped operator running on the Ascend AI Processor
- **RightOp**: name of the operator (running on the GPUs or CPUs) that generates the dump data or .npy file
- **Format**: data format
- **MinAbsoluteError** & **MaxAbsoluteError**: minimum and maximum absolute error ranges
- **MinRelativeError & MaxRelativeError**: minimum and maximum relative error ranges

**Figure 13-9** Single-operator comparison result (details)

<table>
<thead>
<tr>
<th>Index</th>
<th>N</th>
<th>C</th>
<th>H</th>
<th>W</th>
<th>Left</th>
<th>Right</th>
<th>AbsoluteError</th>
<th>RelativeError</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.78418</td>
<td>0.00000</td>
<td>0.78418</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.05468</td>
<td>0.00000</td>
<td>2.05468</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>13.69531</td>
<td>2.80303</td>
<td>7.89103</td>
<td>1.359784</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>6.36328</td>
<td>2.80326</td>
<td>3.56301</td>
<td>1.272392</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>6.36328</td>
<td>2.80303</td>
<td>3.56301</td>
<td>1.272392</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>7.35938</td>
<td>1.65519</td>
<td>5.65074</td>
<td>3.353184</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2.24218</td>
<td>1.75148</td>
<td>2.40930</td>
<td>0.696587</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0.54052</td>
<td>1.75148</td>
<td>0.24093</td>
<td>0.696587</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>3.47656</td>
<td>4.42284</td>
<td>0.94623</td>
<td>0.213953</td>
</tr>
<tr>
<td>16</td>
<td>14</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>-</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>16</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0.01342</td>
<td>0.00000</td>
<td>0.01342</td>
<td>-</td>
</tr>
<tr>
<td>19</td>
<td>17</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3.22851</td>
<td>3.95702</td>
<td>0.72850</td>
<td>0.184104</td>
</tr>
<tr>
<td>20</td>
<td>18</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>5.76562</td>
<td>5.64293</td>
<td>0.12268</td>
<td>0.021742</td>
</tr>
<tr>
<td>21</td>
<td>19</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>0.07788</td>
<td>3.26067</td>
<td>3.18279</td>
<td>0.976115</td>
</tr>
<tr>
<td>22</td>
<td>20</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>-</td>
</tr>
</tbody>
</table>

The detailed comparison result of the single-operator is stored in `{op_name}_input_{index}_{file_index}.csv` or `{op_name}_output_{index}_{file_index}.csv`. Each file records a maximum of one million data records. The parameters in **Figure 13-9** are described as follows:

- **N,C,H,W**: data coordinates.
- **Left**: dump value of the operator running on the Ascend AI Processor.
- **Right**: dump value of the operator running on the GPUs or CPUs.
- **AbsoluteError**: absolute error. The value is the difference between the dump value of the operator in the **Left** column and the dump value of the operator in the **Right** column.
- **RelativeError**: relative error. The value is obtained by dividing the **AbsoluteError** value by the dump value of the operator in the **Right** column. If the dump value of the operator in the **Right** column is 0, a hyphen (-) is displayed.
13.4.5.4 How Do I Convert the Format of a Dump File?

Converting the Format of a Dump File

In the current version, dump files generated by running on the the Ascend AI Processor can be converted into NumPy format.

To perform the conversion, execute the `shape_conversion.py` script stored in the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare` path. The command syntax is as follows:

```
python3.7.5 shape_conversion.py -i DUMP_FILE_PATH -format FORMAT -o OUTPUT_PATH [-shape SHAPE] [-tensor TENSOR] [-index INDEX] [-custom CUSTOM_PATH]
```

The command-line options are described as follows:

- `-i`: dump file (including the path) of a model running on the the Ascend AI Processor.
- `-format`: format of the converted file.
- `-o`: directory of the converted file.
- `-shape`: (optional) shape to be set for FRACTAL_NZ conversion. The shape format is `(\[0-9\]+,)+\[0-9\]+`, where each number must be greater than 0.
- `-tensor`: (optional) indicates whether the dump file to be converted is the input or output. Defaults to `output`.
- `-index`: (optional) sequence number of a tensor, indexed starting at 0.
- `-custom`: (optional) customized path to store the .py file for format conversion, which should be the upper-level directory of the `format_convert` directory. For details about the .py file requirements, see Preparing a Customized .py File for Format Conversion.

**NOTE**

The result is saved in the format of `original_file_name.output.{index}.{shape}.npy` or `original_file_name.input.{index}.{shape}.npy`, where `shape` is formatted as `1x3x224x224`. If the custom format is the same as the built-in format, the custom format applies.

Currently, the following built-in format conversion types are supported:

- FRACTAL_NZ to NCHW
- FRACTAL_NZ to ND
- HWCN to FRACTAL_Z
- HWCN to NCHW
- HWCN to NHWC
- NC1HWC0 to HWCN
- NC1HWC0 to NCHW
- NC1HWC0 to NHWC
- NCHW to FRACTAL_Z
- NCHW to NHWC
- NHWC to FRACTAL_Z
- NHWC to HWCN
- NHWC to NCHW
Preparing a Customized .py File for Format Conversion

Prepare the file as follows:

- The name of the .py file is in `convert_{format_from}_to_{format_to}.py` format. The supported formats for `format_from` and `format_to` are as follows:
  - NCHW
  - NHWC
  - ND
  - NC1HW0
  - FRACTAL_Z
  - NC1C0HWPAD
  - NHWC1C0
  - FSR_NCHW
  - FRACTAL_DECONV
  - C1HWNC0
  - FRACTAL_DECONV_TRANSPOSE
  - FRACTAL_DECONV_SP_STRIDE_TRANS
  - NC1HW0_C04
  - FRACTAL_Z_C04
  - CHWN
  - DECONV_SP_STRIDE8_TRANS
  - NC1KHWHWC0
  - BN_WEIGHT
  - FILTER_HWCK
  - HWCN
  - LOOKUP_LOOKUPS
  - LOOKUP_KEYS
  - LOOKUP_VALUE
  - LOOKUP_OUTPUT
  - LOOKUP_HITS
  - MD
  - NDHW0
  - C1HWNCoC0
  - FRACTAL_NZ

- The content of the .py file is as follows:

```python
def convert(shape_from, shape_to, array):
    return numpy_array
```

The parameters are described as follows:

- `shape_from`: shape of the one-dimensional array before conversion
- `shape_to`: (optional) shape of the one-dimensional array after conversion
- `array`: one-dimensional source data
return: NumPy array after conversion

- The directory of the .py file must meet the following requirement:
The .py file must be stored in the format_convert directory. If the directory
does not exist, create one.

13.4.5.5 How Do I View a Dump File?

Dump files cannot be viewed with a text tool. Therefore, you need to convert your
dump file into a NumPy file and save the NumPy file as a text file using
numpy.savetxt.

Step 1 Log in to the development environment as the installation user.

Step 2 Go to the /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/
operator_cmp/compare directory.

Step 3 Run the dump_data_conversion.py script to convert the dump file into a NumPy
file. The following is an example:

python3.7.5 dump_data_conversion.py -target numpy -type offline -i /home/
HwHiAiUser/dump -o /home/HwHiAiUser/dumptonumpy

NOTE
For details about the parameters of the dump_data_conversion.py script, see
13.4.5.6 How Do I Convert an .npy File into a Dump File?.

Step 4 Use Python to save the NumPy data into a text file. The following is an example.

```python
>>> import numpy as np

>>> a = np.load("/home/HwHiAiUser/dumptonumpy/
Pooling.pool1.1147.158915081588018.output.0.npy")

>>> b = a.flatten()

>>> np.savetxt("/home/HwHiAiUser/dumptonumpy/
Pooling.pool1.1147.158915081588018.output.0.txt", b)
```

The dimension information and Dtype no longer exist in the .txt file. For details,
visit the NumPy website.

---End

13.4.5.6 How Do I Convert an .npy File into a Dump File?

After obtaining an .npy file, upload it to the Ascend AI Processor development
environment and run the dump_data_conversion.py script to convert the .npy file
into a dump file in binary format. The command syntax is as follows:

```bash
python3.7 dump_data_conversion.py -type TYPE -target TARGET -i INPUT_PATH -o OUTPUT_PATH
```

The following is a command example for the conversion:
```bash
python3.7 dump_data_conversion.py -type tf -target dump -i /home/HwHiAiUser/tfnpyfile -o /home/HwHiAiUser/tfdump
```

- **-type:** (required) data type, either `tf` or `offline`
  - `tf`: dump data of a model running on the GPUs or CPUs
  - `offline`: dump data of a model running on the Ascend AI Processor

- **-target:** (required) target format, either `numpy` or `dump`
  - `numpy`: converts a dump file into a NumPy file.
  - `dump`: converts a NumPy file into a dump file.

- **-i:** (required) data folder/file path
  - Converting NumPy files into dump files:
    To pass a folder path to `i`, ensure that names of the files in the folder are in `op_name.output_index.timestamp.npy` format.
    To pass a file path to `i`, ensure that the name of the file is in `op_name.output_index.timestamp.npy` format. Only one file can be set at a time.
    `op_name` must comply with the `A-Za-z0-9_-` regular expression rule, `timestamp` is of 16 bits, and `output_index` is a digit in the range 0–9.

  - Converting dump files into NumPy files:
    To pass a folder path to `i`, ensure that names of the files of network model in the folder are in `op_type.op_name.taskid.timestamp` format, and names of the files of the TensorFlow model in the folder are in `op_name.output_index.timestamp.pb` format.
    To pass a file path to `i`, ensure that the name of the file of the network model is in `op_type.op_name.taskid.timestamp` format, and the name of the file of the TensorFlow model is in `op_name.output_index.timestamp.pb` format. Only one file can be set at a time.
    `op_type` and `op_name` must comply with the `A-Za-z0-9_-` regular expression rule, `timestamp` is of 16 bits, and `taskid` and `output_index` are a digit in the range 0–9.

- **-o:** (required) output path

**NOTE**

You can find the `dump_data_conversion.py` script in `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare`.

To use this script for conversion, ensure that the host has at least 15 GB disk space. If the size of a single dump file to be converted exceeds 441 MB, you are advised to use a host with larger disk space.
14 AI Core Error Analyzer Instructions

14.1 Introduction

Overview

When you encounter AI Core errors in the training process, you can use the AI Core Error Analyzer to collect necessary information for quickly locating the AI Core errors.

Restrictions

- This tool cannot be deployed in a container.
- This tool can be used only for local analysis. That is, the environment where this tool is deployed must be the same as the environment where logs are stored.
14.2 Troubleshooting Workflow

Figure 14-1 Troubleshooting workflow in training scenario

Step 1  Encounter an AI Core error in the model training process.

Step 2  Configure the AI Core Error Analyzer parameters, including:
- `op_debug_level` for locating the code line numbers of error operators.
- `enable_exception_dump` for dumping error operators.

Step 3  Run the model training script again to generate an instruction mapping file and a dump file of each error operator.

Step 4  Analyze the error using the AI Core Error Analyzer.

----End

14.3 Preparing Data

1. If an AI Core error is reported at training time, import the following environment variables before re-training:
   For details, see ATC Instructions and Log Reference.
2. If an AI Core error occurs at training time, perform the following steps to configure the `op_debug_level` and `enable_exception_dump` parameters:

   - In **Estimator** mode, set `op_debug_level` and `enable_exception_dump` as follows.
     ```python
     from npu_bridge.estimator.npu.npu_config import NPURunConfig
     from npu_bridge.estimator.npu.npu_config import DumpConfig
     
     session_config=tf.ConfigProto()
     
     config = NPURunConfig(
         op_debug_level = 2, # Enable operator debug.
         session_config=session_config,
         enable_exception_dump=1 # Dump the inputs and outputs of the error operator to the script execution directory. Dynamic-shape operators cannot be dumped.
     )
     ```

   - In **session.run** mode, set `op_debug_level` and `enable_exception_dump` as follows.
     ```python
     import tensorflow as tf
     from npu_bridge.estimator import npu_ops
     from tensorflow.core.protobuf.rewriter_config_pb2 import RewriterConfig
     
     config = tf.ConfigProto()
     custom_op = config.graph_options.rewrite_options.custom_optimizers.add()
     custom_op.name = "NpuOptimizer"
     custom_op.parameter_map["use_off_line"].b = True
     custom_op.parameter_map["enable_exception_dump"].i = 1 # Dump the inputs and outputs of the error operators to the script execution directory. Dynamic-shape operators cannot be dumped.
     custom_op.parameter_map["op_debug_level"].i = 2 # Enable operator debug.
     config.graph_options.rewrite_options.remapping = RewriterConfig.OFF # Disable remapping.
     
     with tf.Session(config=config) as sess:
         print(sess.run(cost))
     ```

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (default)</td>
<td>Disables operator debug.</td>
</tr>
<tr>
<td>1</td>
<td>Enables operator debug and generates a TBE instruction mapping file. In this case, an operator CCE file (<em>.cce) and a Python-CCE mapping file (</em>.loc.json), and operator .o and .json files are generated in the <code>kernel_meta</code> folder in the training script execution directory. You can locate the AI Core error by using the line numbers in the CCE code and TBE code of the error operator.</td>
</tr>
</tbody>
</table>
Enables operator debug and generates a TBE instruction mapping file. In this case, an operator CCE file (*.cce) and a Python-CCE mapping file (*.loc.json), and operator .o and .json files are generated in the kernel_meta folder in the training script execution directory, and the build optimization is disabled by enabling the CCE compiler `-O0-g`. You can locate the AI Core error by using the line numbers in the CCE code and TBE code of the error operator.

3. Find the instruction mapping file and error operator dump file generated to the training execution directory.

### 14.4 Analyzing AI Core Errors

#### Setting Environment Variables

**NOTE**

This tool supports Python 3.7.0 to Python 3.7.9. This document uses Python 3.7.5 as an example. The environment variables and installation commands vary according to the actual Python version.

Set environment variables as follows. Replace the installation path with the actual one.

```bash
export PATH=/usr/local/python3.7.5/bin:${install_path}/atc/ccec_compiler/bin:${install_path}/atc/bin:$PATH
```

#### Starting AI Core Error Analyzer

**Table 14-2 Command-line options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Short Form</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--compile_path</td>
<td>-c</td>
<td>Yes</td>
<td>Training script execution path.</td>
</tr>
<tr>
<td>--report_path</td>
<td>-p</td>
<td>Yes</td>
<td>Log flush path (named after the timestamp) for log transfer from the device (operating environment) to the host (local environment). For details, see &quot;msnpureport Tool Instructions&quot; in Log Reference.</td>
</tr>
</tbody>
</table>
### Viewing Analysis Result

The outputs of the AI Core Error Analyzer are generated to the `info_xxxx` directory specified by `--output`. The generated result file varies with application cases. The following is an example only.

```
|-- aicore_error
   |-- aicerr_out
      |-- info_xxxx
         |-- aicerr_xxxxx  // Directory of AI Core Error Analyzer outputs
               |-- info.txt  // AI Core Error Analyzer report
               |-- ts.log  // Log file generated after ts.txt analysis in the Black Box
               |-- te_transdata_xxxxx.o  // Decompilation file
               |-- te_transdata_xxxxx.o.txt // Decompilation file
         |-- collection  // Directory of error operator files
               |-- compile
               |   |-- kernel_meta
               |       |-- cce file
               |       |-- json file
               |       |-- loc.json file
               |       |-- o file
               |   |-- ge_proto_xxxxx_Build.txt
               |   |-- dump  // Directory of dump files
               |   |-- log  // Directory of host logs
               |   |-- xxxx  // Timestamp directory
               |       |-- hisi_logs  // Black Box log directory
               |       |-- slog  // slog on the device side
               |       |-- errorLog  // ERROR-level log file
               |       |-- imas.log  // GE IMAS log file
               |       |-- README.txt
         |-- npu_report
            |-- xxxx  // Timestamp directory
            |   |-- hisi_logs  // Black Box log directory
            |   |-- message  // Directory of device-side OS logs
            |   |-- slog
            |   |-- stackcore
```

### 14.5 Locating AI Core Errors

You can locate AI Core errors from the command line or the `info.txt` file.

```
***************Root cause conclusion***************
# Gives the root cause if the error matches known error patterns.
***************1. Basic information***************
```
# Gives the basic information about the device occurred with the AI Core error.
# kernel name: operator kernel name
# op address: address of the operator code in the DDR
# args address: address of the operator arguments in the DDR
error time : 2020-08-26-11:24:07
device id : 0
core id : 0
task id : 60
stream id : 517
node name : trans_TransData_167
kernel name : te_transdata_16b6e15e2a5cc7f70_33e5fb7ae8478ddb
op address : 0x101000120000
args address : 0X101000053000

***************2. AICERROR code***********************
# Gives the AI Core error code and description.
code : 0x10
CCU_ERR_INFO: 0xb166486200070074

***************3. Instructions************************
# Gives the error instructions.
start pc : 0x101000120000
current pc : 0x1010001201e0
Error occured most likely at line: 1d0
/{--output path}/aicerror_xxxx/te_transdata_16b6e15e2a5cc7f70_33e5fb7ae8478ddb.o.txt:1d0
/{--output path}/collection/compile/kernel_meta/te_transdata_16b6e15e2a5cc7f70_33e5fb7ae8478ddb.cce:
  32    //CCE code line number of the error operator
/{Python script path}/nz_2_nd.py:4486 //Python code line number of the error operator
related instructions (error occured before the mark *):
1bc: <not available>
1c0: <not available>
1c4: <not available>
1c8: <not available>
1cc: <not available>
1d0: <not available>
1d4: <not available>
1d8: <not available>
1dc: <not available>
* 1e0: <not available>
For complete instructions, please view /{--output path}/aicerror_xxxx/te_transdata_16b6e15e2a5cc7f70_33e5fb7ae8478ddb.o.txt

***************4. Input and output of node*******************
# Gives the input and output information.
# The input and output addresses are parsed from the IMAS log of GE and the size is parsed from the build graph.
# In the case of memory zero copy, the new address (new addr) can also be parsed from the log.
# If the address is not within the range of the RTS allocation log, an overflow flag is added.
# If the device memory data is collected, NaN and INF verification will also be performed. The collected data is accurate only when the device is suspended.
# If the detected input count and output count are inconsistent with those defined in the kernel function, a WARNING is returned. There is a high probability of misplacement between the arguments provided by GE and those processed by the operator.
input[0]   addr: 0x100801126600  size: 32288
output[0]  addr: 0x100801157c00  size: 2048

***************5. Op in graph******************************
# Gives information of the error operator.
# The operator information is taken from the build graph for viewing convenience.

***************6. Dump info*******************************
# Dump file of the error operator.
# This information is available only in the training scenario.