Model Conversion Guide

Issue 01

Date 2020-05-30

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

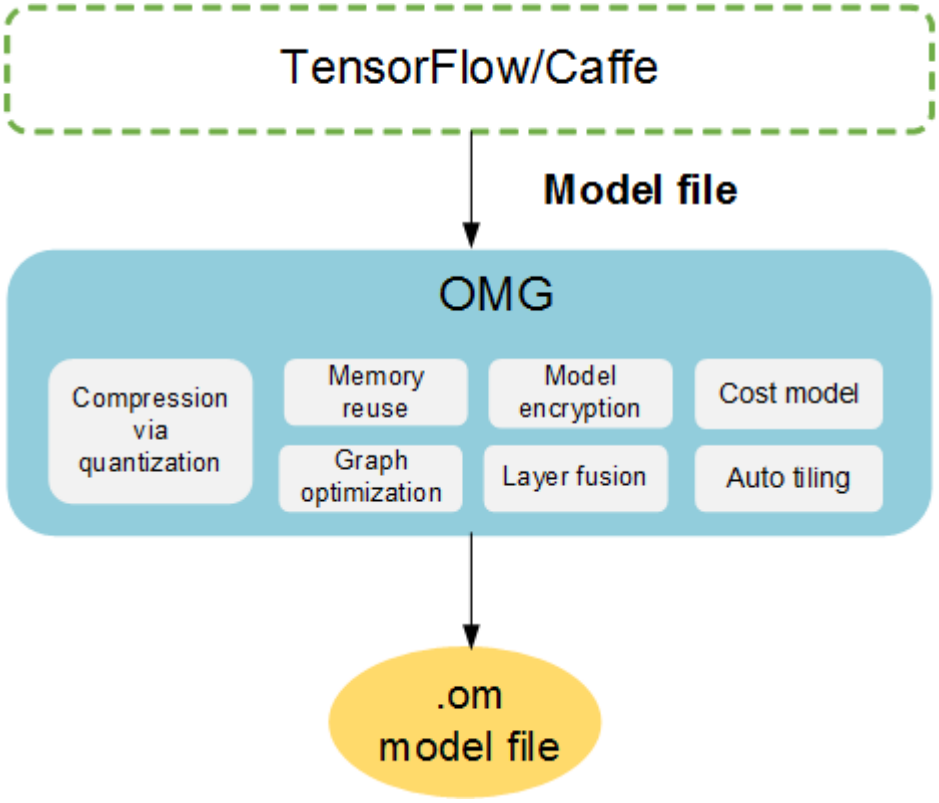
1 Introduction

Trained models under frameworks such as Caffe and TensorFlow can be converted into offline models supported by the Ascend AI processor by using the offline model generator (OMG). During offline model conversion, you can enable operator scheduling optimization, data re-orchestration in the weight file, model compression via quantization, and memory usage optimization, thereby pre-processing the model without depending on the device.

Functional Architecture

Figure 1-1 shows the functional architecture of OMG.

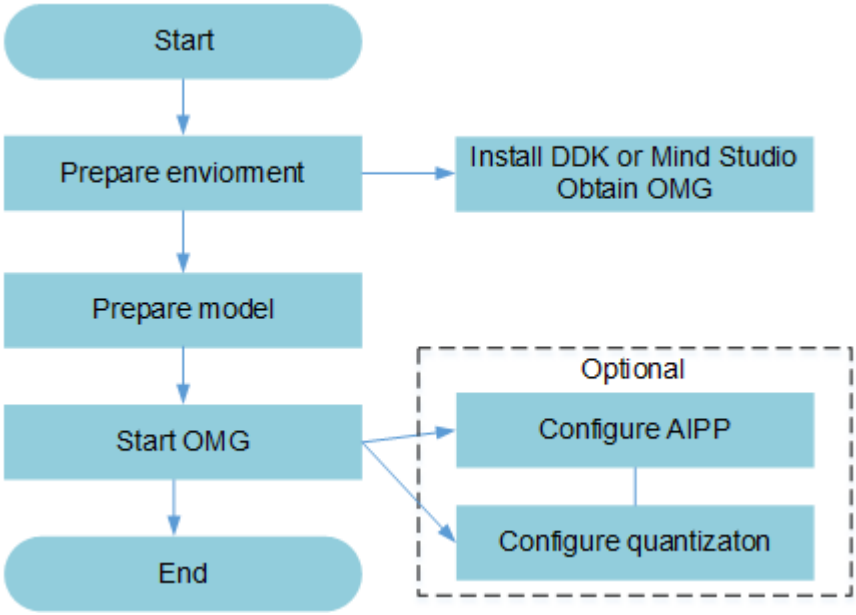
Figure 1-1 OMG functional architecture



OMG Workflow

Figure 1-2 shows the model conversion workflow of OMG.

Figure 1-2 OMG workflow



The OMG workflow is described as follows:

1. Before using OMG, install the DDK or the Mind Studio IDE on the server and obtain the OMG tool from the specified path. For details, see **Preparations**.
2. Upload the model to be converted to the server where the DDK is located. For details, see **Sample**.
3. Start OMG to convert the model. Enable artificial intelligence pre-processing (AIPP) or quantization as required. For details, see **3 AIPP Configuration** and **4 Quantization Configuration**.
    a. The Ascend AI processor introduces the AIPP module for hardware-based image pre-processing including color space conversion (CSC), image normalization (by subtracting the mean value or multiplying a factor), image cropping (by specifying the crop start and cropping the image to the size required by the neural network), and much more. Images output by the DVPP module are aligned images in YUV420SP format. The digital vision pre-processing (DVPP) module does not output RGB images. Therefore, the AIPP module is introduced to convert the aligned YUV420SP images and then crop them to the size required by the model.
    b. Quantization refers to low-bit quantization of high-precision data, suitable for scenarios that are model-size and performance demanding. This feature results in lightweight models, with reduced storage spaces and transfer latency but improved computation efficiency.

2 Caffe/TensorFlow Mode Conversion

This topic describes how to use the OMG tool to convert a Caffe or TensorFlow model to an offline model supported by the Ascend AI processor.

2.1 Constraints and Parameters

Restrictions

Before model conversion, pay attention to the following restrictions:

- Only a Caffe or TensorFlow model can be converted. For a Caffe model, the input data must be of the FLOAT type. For a TensorFlow model, the input data must be of INT32, BOOL, UINT8, or FLOAT type.
- For a Caffe model, **op name** and **op type** in the model file (.prototxt) and weight file (.caffemodel) must be the consistent (case sensitive).
- For a Caffe model, the **top** names of all layers must be the same except for the layers with the same **top** and **bottom** (such as BatchNorm, Scale, and ReLU).
- For a TensorFlow model, only the FrozenGraphDef format is supported.
- Inputs with dynamic shapes are not supported, for example, NHWC = [?, ?, ?, 3]. The dimension sizes must be static.
- The input can be up to 4-dimensional. Operators involving dimension changes (such as reshape and expanddim) cannot output five dimensions.
- Except the const operator, the input and output of operators at all layers in the model must meet the requirements of **dim! = 0**.
- Model conversion does not support models that contain operators used for model training.
- A UINT8 quantized model cannot be converted.
- Model operators support only 2D convolution, and do not support 3D convolution.
- Only the operators listed in *Operator List* are supported. The defined restrictions on the operators must be met.

Parameter Description

Parameter	Description	Mandatory or Not (Depending on Whether the Value of Mode Is 0 or 3)	Default Value
--mode	Operating mode ● 0: Generate an offline model supported by the Ascend AI processor ● 1: Convert the offline model or model file to the JSON format. ● 3: Perform precheck only to check the validity of the model file.	No	0
--model	Path of the source model file **NOTE** The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	Yes	N/A
--weight	Path of the weight file This parameter needs to be specified when the source model framework is Caffe. **NOTE** The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	No	N/A
--framework	Framework of the source model ● 0: Caffe ● 3: TensorFlow **NOTE** ● This parameter is not mandatory when mode is set to **1**. If this parameter is not set, the offline model is converted to the JSON format by default (ensure that **--om** and **--framework** are consistent). --framework=0 --om=/home/*username*/*test*/resnet18.prototxt ● This parameter is mandatory when **mode** is set to **0** or **3**.	Yes	N/A

Parameter	Description	Mandatory or Not (Depending on Whether the Value of Mode Is 0 or 3)	Default Value
--output	Path for storing the converted offline model (including the file name), for example, **out/caffe_resnet18**. The converted offline model is automatically suffixed with **.om**. **NOTE** The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	Yes	N/A
--encrypt_mode (reserved)	Encryption mode ● 0: encrypted ● -1: not encrypted	No	-1
--encrypt_key (reserved)	Path of the random number file used for encryption This parameter is mandatory in encryption mode. **NOTE** ● The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.). ● During the test, you can run the **openssl rand 32 -out ek_key** command to generate a random number file. In commercial use, a random number file can be generated by using other tools as required.	No	N/A
--hardware_key (reserved)	Path of the encrypted ISV hardware key file This parameter is mandatory in encryption mode. **NOTE** The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	No	N/A

Parameter	Description	Mandatory or Not (Depending on Whether the Value of Mode Is 0 or 3)	Default Value
--certificate (reserved)	Path of the ISV certificate file used for encryption This parameter is mandatory in encryption mode. **NOTE** The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	No	N/A
--private_key (reserved)	Path of the ISV private key file used for encryption This parameter is mandatory in encryption mode. **NOTE** The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	No	N/A
--cal_conf	Path of the quantization configuration file **NOTE** • The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.). • The following is an example of the quantization configuration file: device: USE_CPU bin: 150 type: JSD quantize_algo: NON_OFFSET inference_with_data_quantized: true inference_with_weight_quantized: true • For details about the quantization configuration file, see **4 Quantization Configuration**.	No	N/A

Parameter	Description	Mandatory or Not (Depending on Whether the Value of Mode Is 0 or 3)	Default Value
--check_report	Path of the precheck result file. If this path is not specified, the precheck result is saved in the current path when the model conversion fails or **mode** is set to **3** (precheck only). **NOTE** The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	No	check_result.json
--h or --help	Help information	No	N/A
--input_format	Input data format, either **NCHW** or **NHWC** ● For TensorFlow, the default value is **NHWC**. To use the NCHW format, you need to specify **NCHW**. ● For Caffe, only **NCHW** is supported.	No	N/A
--input_fp16_nodes	This parameter is used in conjunction with the **--is_output_fp16** parameter. Name of the FP16 input node to the second network in the event of network cascade Example: **node_name1;node_name2** For example: Two networks net1 and net2 are cascaded. The output of net1 serves as the input to net2. This parameter is used to specify the name of the FP16 input node to the second network in the event of network cascade.	No	N/A

Parameter	Description	Mandatory or Not (Depending on Whether the Value of Mode Is 0 or 3)	Default Value
--input_shape	Shape of the input data Example: **input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2** **input_name** must be the node name in the network model before model conversion. If the source model is of a dynamic shape, for example, **input_name1:? ,h,w,c**. This parameter is mandatory. Replace **?** with the actual batch size. It is used to convert the source model with a dynamic shape into a offline model with a fixed shape.	No	N/A
--is_output_fp16	Whether the output data type of the first network is FP16 in the event of network cascade For example: **false,true,false,true** For example: Two networks net1 and net2 are cascaded. The output of net1 serves as the input to net2. This parameter is used to specify the output data type of net1 as FP16.	No	false
--json	Path of the .json file converted from the offline model **NOTE** The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	No	N/A

Parameter	Description	Mandatory or Not (Depending on Whether the Value of Mode Is 0 or 3)	Default Value
--om	This parameter is mandatory when **mode** is set to **1**. Path of the offline model or model file to be converted to the JSON format For example: **/home/*username*/test/out/caffe_resnet18.om** or **/home/*username*/test/resnet18.prototxt** **NOTE** The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	No	N/A
--op_name_map	Path of the operator mapping configuration file. This parameter must be specified when the DetectionOutput algorithm is used on the network. For example, the DetectionOutput operator can play different roles in different networks. It can specify the mapping from DetectionOutput (of a Da Vinci model) to the following operators: ● FSRDetectionOutput (of the Faster R-CNN network) ● SSDDetectionOutput (of the SSD network) ● RefinedetDetectionOutput (of the RefineDet network) **NOTE** ● The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.). ● The following is an example of the operator mapping configuration file: **DetectionOutput: SSDDetectionOutput**	No	N/A

Parameter	Description	Mandatory or Not (Depending on Whether the Value of Mode Is 0 or 3)	Default Value
--out_nodes	Output node If the output node (operator name) is not specified, the output of the last operator layer serves as the model output by default. To check the parameters of a specific operator layer, specify the operator layer by using this parameter. After the model is converted, you can view the parameter information of the specified operator at the end of the .om model file or the JSON file converted from the .om model file. Example: **node_name1:0;node_name1:1;node_name2:0** **node_name** must be the node name in the network model before model conversion. The number after the colon indicates the output index. For example, **node_name1:0** indicates output 0 of the node named **node_name1**.	No	N/A
--plugin_path	Paths of the custom operator plug-ins Example: **/home/a1/b1;/home/a2/b2;/home/a3/b3** **NOTE** Separate multiple paths by semicolons (;). However semicolons (;) are not allowed in a path; otherwise, path parsing fails.	No	./plugin
--target	This parameter can only be set to **mini**. **mini**: The eltwise operator in quantization supports dual outputs. The roipooling operator in quantization supports int8 output. The conv operator in quantization supports hybrid precision.	No	mini
--ddk_version	Version of the DDK environment to be matched for running a custom operator	No	N/A

Parameter	Description	Mandatory or Not (Depending on Whether the Value of Mode Is 0 or 3)	Default Value
--net_format	Preferred data format for network operators: **ND** (N ≤ 4) or **5D**. This parameter is valid only when the input data of the operator on the network supports both the ND and 5D formats. • **ND**: The operators in the model are converted into the common format NCHW. • **5D**: The operators in the model are converted into Huawei-developed format 5D.	No	N/A

--insert_op_conf	Configuration file of the preprocessing operator, for example, aipp operator
	NOTE
	• The path can contain uppercase letters, lowercase letters, digits, and underscores (_). The file name can contain uppercase letters, lowercase letters, digits, underscores (_), and periods (.).
	• The following is an example of the configuration file:
	aipp_op {
	aipp_mode: static
	input_format: YUV420SP_U8
	csc_switch: true
	var_reci_chn_0: 0.00392157
	var_reci_chn_1: 0.00392157
	var_reci_chn_2: 0.00392157
	}
	• For details about the AIPP configuration file, see **3 AIPP Configuration**.
		No	N/A

Parameter	Description	Mandatory or Not (Depending on Whether the Value of Mode Is 0 or 3)	Default Value
--fp16_high_prec	Whether to generate a high-accuracy FP16 Da Vinci model. ● **0** (default): Generate a common FP16 Da Vinci model, which shows better inference performance. ● **1**: Generate a high-accuracy FP16 Da Vinci model, which offers higher inference accuracy. High-precision model generation supports only the following operators: Convolution, Pooling, and FullConnection of Caffe and tf.nn.conv2d and tf.nn.max_pool of TensorFlow.	No	0
--output_type	Network output data type ● **FP32** (default): It is recommended for classification and detection networks. ● **UINT8**: It is recommended for image super-resolution networks for better inference performance.	No	FP32
--enable_l2dynamic	L2 dynamic optimization enable. This parameter may affect the inference performance of the network model. If the performance does not meet the requirement, you can disable this switch to verify the impact on the performance. ● **true**: L2 dynamic optimization enabled ● **false**: L2 dynamic optimization disabled	No	true

2.2 Model Conversion Using OMG

Preparations

- **Environment preparation**

  This document describes how to convert models in command line interface (CLI) mode. For details about how to convert models in the Mind Studio IDE, see *Ascend 310 Mind Studio Basic Operations*.

The following describes the scenario where the DDK is independently installed.

If the DDK installation user is **HwHiAiUser** and the log fails to be output to the screen, see **5.1 What Do I Do If No Log Is Output to the Screen When the DDK Installation User Is HwHiAiUser?**. If the OS and architecture configuration of the DDK server is Arm (aarch64) and model conversion takes too long, see **5.2 What Do I Do If Model Conversion Takes Too Long When the OS and Architecture Configuration of the DDK Server is Arm (aarch64)?**.

- Upload the model file (*.prototxt), weight file (*.caffemodel) required by model conversion to the **/home/*username*/test/** directory on the DDK server as the DDK installation user.

Sample

Step 1 Log in to the DDK server as the DDK installation user.

Step 2 Set the following environment variable:

```
export LD_LIBRARY_PATH=DDKInstallationDirectory/ddk/uihost/lib
```

Step 3 Find OMG in the **ddk/uihost/bin** directory under the DDK installation directory. Run the following command to generate a model file. (The directories and files in the command are for reference only.)

```
./omg --model=/home/username/test/resnet18.prototxt --weight=/home/username/test/
resnet18.caffemodel --framework=0 --output=/home/username/test/out/caffe_resnet18
```

After the command is executed successfully, you can view the model file (for example, **caffe_resnet18.om**).

NOTE

A command that exceeds one line will be automatically wrapped due to the restriction of the PDF document format. Therefore, if you want to use the command in the example directly, you need to manually merge the lines into one line and separate the parameters with spaces.

For details about how to fix the following errors during model conversion (**It is recommended to convert layers-structure to layer-structure by caffe tool** and **Type XXX unsupported**), see **5 FAQs**.

Step 4 (Optional) If an output node is specified (that is, the **--out_nodes** parameter is set) during model conversion and the information of the last operator layer fails to be viewed in the converted .om model, convert the .om model file to the JSON format by running the following command and check the .json file:

```
omg --mode=1 --om=/home/username/test/caffe_resnet18.om  --json=/home/username/test/out/resnet.json
```

----End

3 AIPP Configuration

3.1 Overview

AIPP involves image resizing, CSC, and mean value subtraction and factor multiplication (for pixel changing). All these functions are implemented by the AI Core.

Static AIPP and dynamic AIPP modes are supported. However, the two modes are mutually exclusive.

- Static AIPP: During model conversion, set the AIPP mode to static and set the AIPP parameters. After the Da Vinci model is generated, the AIPP parameter values are saved in the model. The same AIPP parameter configurations are used in each model inference phase.

  If the static AIPP mode is used, batches share the same set of AIPP parameters.
- Dynamic AIPP: During model conversion, set the AIPP mode to dynamic. The AIPP parameters must be set in the code of the inference engine before each model inference. In this way, different sets of AIPP configurations can be used for different model inference phases. For details about the dynamic AIPP APIs and usage example, see **AIPP Configuration APIs** in *Matrix API Reference*.

  In dynamic AIPP mode, batches use separate parameter configurations (such as the crop and resize attributes) defined by the dynamic parameter structure. For details about the dynamic parameter structure, see **3.7 Parameter Structure for Dynamic AIPP**.

AIPP supports the following image input formats: **YUV420SP_U8**, **XRGB8888_U8**, **RGB888_U8**, and **YUV400_U8**.

- YUV420SP_U8 is classified into YUV420SP_UV(NV12) and YUV420SP_VU(NV21) based on the UV sequence. The default format is YUV420SP_UV(NV12).
  - If **rbuv_swap_switch** in **3.2 Configuration File Template** is set to **false**, the AIPP output format is YUV420SP_UV(NV12)
  - If **rbuv_swap_switch** in **3.2 Configuration File Template** is set to **true**, the AIPP output format is YUV420SP_UV(NV12).
- For RGB888_U8, the AIPP output format varies according to the value of **rbuv_swap_switch**.
  - If **rbuv_swap_switch** in **3.2 Configuration File Template** is set to **false**, the AIPP output format is RGB888_U8.
  - If **rbuv_swap_switch** in **3.2 Configuration File Template** is set to **true**, the AIPP output format is BGR888_U8.

NOTICE

- The input format of AIPP is **YUV420SP_U8** (the default format is **YUV420SP_UV**). If the format is **YUV420SP_VU**, set **rbuv_swap_switch** to **false**. Otherwise, the output result will be affected.
- When AIPP is enabled, the model input is in **RGB888_U8** or **BGR888_U8** format. The two formats correspond to different CSC matrices.
- Pay attention to the following requirements on the input images:
  - If AIPP is enabled for model conversion and the input image format is **XRGB8888_U8** or **RGB888_U8**, the converted model accepts NHWC input only.
  - If AIPP is disabled for model conversion, the converted model accepts NCHW input only for inference. Therefore, you need to manually convert the NHWC data to NCHW data.

3.2 Configuration File Template

The following describes the configuration file.

```
# The AIPP configuration starts with the aipp_op flag, which indicates that it is the configuration of an AIPP operator. Multiple aipp_op flags can be configured.
aipp_op {
#
# AIPP provides the following features: CSC, cropping, mean subtraction, multiplication factor, data exchange between channels, and single-line mode.
# The input images must in the UINT8 data type.
# To use this configuration file, delete the comments of parameters to be configured and set the parameter values as required.
# The parameter values in the template are default values. The input_format attribute is mandatory, while other attributes are optional.
#======================== Global settings (start)
================================================================================================================================
# aipp_mode specifies the AIPP mode. This parameter is mandatory.
# Type: enum
```



```
# Value range: dynamic or static (dynamic indicates dynamic AIPP, and static indicates static AIPP.)
# aipp_mode:

# related_input_rank is optional. It indicates the processing start of the inputs. The default value is 0, which indicates that AIPP processing starts from input 0. For example, if the model has two inputs, to have the AIPP processing start from the second input, set related_input_rank to 1.
# Type: int
# Value range: ≥ 0
# related_input_rank: 0

# input_edge_idx is optional. If a model input is shared by multiple operators, that is, the data operator is followed by more operators, set this parameter to perform different AIPP processing for different output dimensions of the Data operator.
# Type: int
# Value range: ≥ 0
# input_edge_idx: 0
#======================== Global settings (end)
================================================================================================================================

#======================== Dynamic AIPP settings (start)
================================================================================================================================
# Maximum size of the input image (must be greater than or equal to that of the source image)
# Type: int
# max_src_image_size: 0

# Whether to enable rotation. This parameter is reserved. Rotation is not supported currently.
# Type: bool
# Value range: true (yes) or false (no)
# support_rotation: false
#======================== Dynamic AIPP settings (end)
================================================================================================================================

#======================== Static AIPP settings (start)
================================================================================================================================
# Input image format
# Type: enum
# Value: YUV420SP_U8, XRGB8888_U8, RGB888_U8, or YUV400_U8
# input_format :

# Image width and height
# Type: uint13
# Value range: [0, 4096]. For YUV420SP_U8 images, the value must be an even number.
# Note: Set src_image_size_w and src_image_size_h to the actual image width and height respectively. If they are not set or set to 0, the width and height defined in the network input are used.
# src_image_size_w: 0
# src_image_size_h: 0

# c_padding_value: 0.0
#========= Crop settings (For the configuration example, see section AIPP Configuration > Crop/Padding Configuration.) =========
# Whether to enable crop in AIPP
# Type: bool
# Value range: true (yes) or false (no)
# crop :false

# Horizontal and vertical coordinates of the crop start. W and H defined in the network input are used as the size of the cut out image.
# Type: uint13
# Value range: [0, 4096]. For YUV420SP_U8 images, the value must be an even number.
# Note: load_start_pos_w plus W defined in the network input must be less than or equal to src_image_size_w.  load_start_pos_h plus H defined in the network input must be less than or equal to src_image_size_h.
# load_start_pos_w: 0
# load_start_pos_h: 0
```



```
# Size of the cut out image
# Type: uint13
# Value range: even number within [0, 4096], load_start_pos_w + crop_size_w ≤ src_image_size_w, and
load_start_pos_h + crop_size_h ≤ src_image_size_h
# crop_size_w: 0
# crop_size_h: 0


#======================== Resize settings ========================
# Whether to enable resize. This parameter is reserved. Rotation is not supported currently.
# Type: bool
# Value range: true (yes) or false (no)
resize :false

# Width and height of the resized image. This parameter is reserved. Resize is not supported currently.
# Type: uint13
# Value range: even number within [0, 4096], less than src_image_size
resize_output_w: 0
resize_output_h: 0


#========= Padding settings (For the configuration example, see section AIPP Configuration > Crop/
Padding Configuration.) =========
# Whether to enable padding in AIPP
# Type: bool
# Value range: true (yes) or false (no)
# padding: false

# Padding in the C direction, for static AIPP
# Type: float16
# left_padding_size: 0
# right_padding_size: 0
# top_padding_size: 0
# bottom_padding_size: 0


#================================ Rotation settings ================================
# Rotation angle. This parameter is reserved. Rotation is not supported currently.
# Type: uint8
# Value range: {0, 1, 2, 3}, where, 0 indicates zero rotation, 1 indicates 90° clockwise rotation, 2 indicates
180° clockwise rotation, and 3 indicates 270° clockwise rotation.
# rotation_angle :0


#========= CSC settings (For the configuration example, see section AIPP Configuration > CSC
Configuration.) =============
# Whether to enable CSC in static AIPP
# Type: bool
# Value: true or false. The value true indicates enabled, and the value false indicates disabled.
# csc_switch :false

# Whether to enable R/B or U/V channel swap before CSC
# Type: bool
# Value: true or false. The value true indicates enabled, and the value false indicates disabled.
# rbuv_swap_switch: false

# Whether to enable single line mode (in this mode, only the first line of the cut out image is processed)
# Type: bool
# Value: true or false. The value true indicates enabled, and the value false indicates disabled.
# single_line_mode: false

# If the CSC is disabled (false), this function is bypassed.
# If the input image has four channels, the first channel is ignored.
# YUV2BGR conversion:
# | B |   | matrix_r0c0 matrix_r0c1 matrix_r0c2 | | Y - input_bias_0 |
# | G | = | matrix_r1c0 matrix_r1c1 matrix_r1c2 | | U - input_bias_1 | >> 8
# | R |   | matrix_r2c0 matrix_r2c1 matrix_r2c2 | | V - input_bias_2 |
# BGR2YUV conversion:
# | Y |   | matrix_r0c0 matrix_r0c1 matrix_r0c2 | | B |        | output_bias_0 |
```

```
# | U | = | matrix_r1c0 matrix_r1c1 matrix_r1c2 | | G | >> 8 + | output_bias_1 |
# | V |   | matrix_r2c0 matrix_r2c1 matrix_r2c2 | | R |        | output_bias_2 |

# Elements in a 3 x 3 CSC matrix
# Type: int16
# Value range: [–32768, +32767]
# matrix_r0c0 :298
# matrix_r0c1 :516
# matrix_r0c2 :0
# matrix_r1c0 :298
# matrix_r1c1 :-100
# matrix_r1c2 :-208
# matrix_r2c0 :298
# matrix_r2c1 :0
# matrix_r2c2 :409

# Output bias for RGB2YUV conversion
# Type: uint8
# Value range: [0, 255]
# output_bias_0: 16
# output_bias_1: 128
# output_bias_2: 128

# Input bias for YUV2RGB conversion
# Type: uint8
# Value range: [0, 255]
# input_bias_0: 16
# input_bias_1: 128
# input_bias_2: 128


#============================== Mean subtraction and multiplication factor settings
==============================
# The computation rules are as follows:
# For uint8->uint8, this function is bypassed.
# For uint8->int8: pixel_out_chx(i) = pixel_in_chx(i) – mean_chn_i
# For uint8->fp16: pixel_out_chx(i) = [pixel_in_chx(i) – mean_chn_i – min_chn_i] * var_reci_chn

# Mean value of channel n
# Type: uint8
# Value range: [0, 255]
# mean_chn_0: 0
# mean_chn_1: 0
# mean_chn_2: 0

# Minimum value of channel n
# Type: float16
# Value range: [–65504, +65504]
# min_chn_0 :0.0
# min_chn_1 :0.0
# min_chn_2 :0.0

# Standard deviation of channel n or reciprocal of (Max. – Min.)
# Type: float16
# Value range: [–65504, +65504]
# var_reci_chn_0: 1.0
# var_reci_chn_1: 1.0
# var_reci_chn_2: 1.0
}
#======================== Static AIPP settings (end)
================================================================================
======
```

3.3 CSC Configuration

After confirming the image formats before and after AIPP (that is, the image format of the model input and image format during model training), you can

determine the CSC parameter values. (The value of **matrix_r*c*** is fixed and does not need to be adjusted.) The following describes some configuration references.

Before AIPP, the following two typical CSC configurations are provided for the images or videos (in various color encoding modes such as YUV420SP_U8, XRGB8888_U8, RGB888_U8, and YUV400_U8) input to the model:

- For JPEG image files (such as JPG, JPEG, JPG, and JPEG), you can select from the CSC configurations in the JPEG columns in the following tables.
- For the decoded video data, the CSC parameters can be configured according to different color video digital standards (such as BT.601 and BT.709).

YUV420SP_U8 to YVU444SP_U8

```
aipp_op {
    aipp_mode: static
    input_format : YUV420SP_U8
    csc_switch : false
    rbuv_swap_switch : true
}
```

YUV420SP_U8 to RGB888_U8

Table 3-1 YUV420SP_U8 to RGB888_U8

JPEG	BT-601NARROW	BT-709NARROW

aipp_op {
aipp_mode: static
input_format : YUV420SP_U8
csc_switch : true
rbuv_swap_switch : false
matrix_r0c0 : 256
matrix_r0c1 : 0
matrix_r0c2 : 359
matrix_r1c0 : 256
matrix_r1c1 : -88
matrix_r1c2 : -183
matrix_r2c0 : 256
matrix_r2c1 : 454
matrix_r2c2 : 0
input_bias_0 : 0
input_bias_1 : 128
input_bias_2 : 128
}
	aipp_op {
	aipp_mode: static
	input_format : YUV420SP_U8
	csc_switch : true
	rbuv_swap_switch : false
	matrix_r0c0 : 298
	matrix_r0c1 : 0
	matrix_r0c2 : 409
	matrix_r1c0 : 298
	matrix_r1c1 : -100
	matrix_r1c2 : -208
	matrix_r2c0 : 298
	matrix_r2c1 : 516
	matrix_r2c2 : 0
	input_bias_0 : 16
	input_bias_1 : 128
	input_bias_2 : 128
	}
		aipp_op {
		aipp_mode: static
		input_format : YUV420SP_U8
		csc_switch : true
		rbuv_swap_switch : false
		matrix_r0c0 : 298
		matrix_r0c1 : 0
		matrix_r0c2 : 459
		matrix_r1c0 : 298
		matrix_r1c1 : -55
		matrix_r1c2 : -136
		matrix_r2c0 : 298
		matrix_r2c1 : 541
		matrix_r2c2 : 0
		input_bias_0 : 16
		input_bias_1 : 128
		input_bias_2 : 128
		}

YUV420SP_U8 to BGR888_U8

Table 3-2 YUV420SP_U8 to BGR888_U8

JPEG	BT-601NARROW	BT-709NARROW

aipp_op {
aipp_mode: static
input_format : YUV420SP_U8
csc_switch : true
rbuv_swap_switch : false
matrix_r0c0 : 256
matrix_r0c1 : 454
matrix_r0c2 : 0
matrix_r1c0 : 256
matrix_r1c1 : -88
matrix_r1c2 : -183
matrix_r2c0 : 256
matrix_r2c1 : 0
matrix_r2c2 : 359
input_bias_0 : 0
input_bias_1 : 128
input_bias_2 : 128
}
	aipp_op {
	aipp_mode: static
	input_format : YUV420SP_U8
	csc_switch : true
	rbuv_swap_switch : false
	matrix_r0c0 : 298
	matrix_r0c1 : 516
	matrix_r0c2 : 0
	matrix_r1c0 : 298
	matrix_r1c1 : -100
	matrix_r1c2 : -208
	matrix_r2c0 : 298
	matrix_r2c1 : 0
	matrix_r2c2 : 409
	input_bias_0 : 16
	input_bias_1 : 128
	input_bias_2 : 128
	}
		aipp_op {
		aipp_mode: static
		input_format : YUV420SP_U8
		csc_switch : true
		rbuv_swap_switch : false
		matrix_r0c0 : 298
		matrix_r0c1 : 541
		matrix_r0c2 : 0
		matrix_r1c0 : 298
		matrix_r1c1 : -55
		matrix_r1c2 : -136
		matrix_r2c0 : 298
		matrix_r2c1 : 0
		matrix_r2c2 : 459
		input_bias_0 : 16
		input_bias_1 : 128
		input_bias_2 : 128
		}

YUV420SP_U8 to Gray

```
aipp_op {
    aipp_mode: static
    input_format : YUV420SP_U8
    csc_switch : true
    rbuv_swap_switch : false
    matrix_r0c0 : 256
    matrix_r0c1 : 0
    matrix_r0c2 : 0
    matrix_r1c0 : 0
    matrix_r1c1 : 0
    matrix_r1c2 : 0
    matrix_r2c0 : 0
    matrix_r2c1 : 0
    matrix_r2c2 : 0
    input_bias_0 : 0
    input_bias_1 : 0
    input_bias_2 : 0
}
```


XRGB8888_U8 to YUV444SP_U8

Table 3-3 XRGB8888_U8 to YUV444SP_U8

JPEG	BT-601NARROW	BT-709NARROW

aipp_op {
aipp_mode: static
input_format : XRGB8888_U8
csc_switch : true
rbuv_swap_switch : false
matrix_r0c0 : 77
matrix_r0c1 : 150
matrix_r0c2 : 29
matrix_r1c0 : -43
matrix_r1c1 : -85
matrix_r1c2 : 128
matrix_r2c0 : 128
matrix_r2c1 : -107
matrix_r2c2 : -21
output_bias_0 : 0
output_bias_1 : 128
output_bias_2 : 128
}
	aipp_op {
	aipp_mode: static
	input_format : XRGB8888_U8
	csc_switch : true
	rbuv_swap_switch : false
	matrix_r0c0 : 66
	matrix_r0c1 : 129
	matrix_r0c2 : 25
	matrix_r1c0 : -38
	matrix_r1c1 : -74
	matrix_r1c2 : 112
	matrix_r2c0 : 112
	matrix_r2c1 : -94
	matrix_r2c2 : -18
	output_bias_0 : 16
	output_bias_1 : 128
	output_bias_2 : 128
	}
		aipp_op {
		aipp_mode: static
		input_format : XRGB8888_U8
		csc_switch : true
		rbuv_swap_switch : false
		matrix_r0c0 : 47
		matrix_r0c1 : 157
		matrix_r0c2 : 16
		matrix_r1c0 : -26
		matrix_r1c1 : -87
		matrix_r1c2 : 112
		matrix_r2c0 : 112
		matrix_r2c1 : -102
		matrix_r2c2 : -10
		output_bias_0 : 16
		output_bias_1 : 128
		output_bias_2 : 128
		}

XRGB8888_U8 to YVU444SP_U8

Table 3-4 XRGB8888_U8 to YVU444SP_U8

JPEG	BT-601NARROW	BT-709NARROW

aipp_op {
aipp_mode: static
input_format : XRGB8888_U8
csc_switch : true
rbuv_swap_switch : false
matrix_r0c0 : 77
matrix_r0c1 : 150
matrix_r0c2 : 29
matrix_r1c0 : 128
matrix_r1c1 : -107
matrix_r1c2 : -21
matrix_r2c0 : -43
matrix_r2c1 : -85
matrix_r2c2 : 128
output_bias_0 : 0
output_bias_1 : 128
output_bias_2 : 128
}
	aipp_op {
	aipp_mode: static
	input_format : XRGB8888_U8
	csc_switch : true
	rbuv_swap_switch : false
	matrix_r0c0 : 66
	matrix_r0c1 : 129
	matrix_r0c2 : 25
	matrix_r1c0 : 112
	matrix_r1c1 : -94
	matrix_r1c2 : -18
	matrix_r2c0 : -38
	matrix_r2c1 : -74
	matrix_r2c2 : 112
	output_bias_0 : 16
	output_bias_1 : 128
	output_bias_2 : 128
	}
		aipp_op {
		aipp_mode: static
		input_format : XRGB8888_U8
		csc_switch : true
		rbuv_swap_switch : false
		matrix_r0c0 : 47
		matrix_r0c1 : 157
		matrix_r0c2 : 16
		matrix_r1c0 : 112
		matrix_r1c1 : -102
		matrix_r1c2 : -10
		matrix_r2c0 : -26
		matrix_r2c1 : -87
		matrix_r2c2 : 112
		output_bias_0 : 16
		output_bias_1 : 128
		output_bias_2 : 128
		}

XRGB8888_U8 to Gray

```
aipp_op {
    aipp_mode: static
    input_format : XRGB8888_U8
    csc_switch : true
    rbuv_swap_switch : false
    matrix_r0c0 : 76
    matrix_r0c1 : 150
    matrix_r0c2 : 30
```



```
    matrix_r1c0 : 0
    matrix_r1c1 : 0
    matrix_r1c2 : 0
    matrix_r2c0 : 0
    matrix_r2c1 : 0
    matrix_r2c2 : 0
    output_bias_0 : 0
    output_bias_1 : 0
    output_bias_2 : 0
}
```

RGB888_U8 to BGR888_U8

```
aipp_op {
    aipp_mode: static
    input_format : RGB888_U8
    csc_switch : false
    rbuv_swap_switch : true
}
```

RGB888_U8 to YUV444SP_U8

Table 3-5 RGB888_U8 to YUV444SP_U8

JPEG	BT-601NARROW	BT-709NARROW

aipp_op {
aipp_mode: static
input_format : RGB888_U8
csc_switch : true
rbuv_swap_switch : false
matrix_r0c0 : 77
matrix_r0c1 : 150
matrix_r0c2 : 29
matrix_r1c0 : -43
matrix_r1c1 : -85
matrix_r1c2 : 128
matrix_r2c0 : 128
matrix_r2c1 : -107
matrix_r2c2 : -21
output_bias_0 : 0
output_bias_1 : 128
output_bias_2 : 128
}
	aipp_op {
	aipp_mode: static
	input_format : RGB888_U8
	csc_switch : true
	rbuv_swap_switch : false
	matrix_r0c0 : 66
	matrix_r0c1 : 129
	matrix_r0c2 : 25
	matrix_r1c0 : -38
	matrix_r1c1 : -74
	matrix_r1c2 : 112
	matrix_r2c0 : 112
	matrix_r2c1 : -94
	matrix_r2c2 : -18
	output_bias_0 : 16
	output_bias_1 : 128
	output_bias_2 : 128
	}
		aipp_op {
		aipp_mode: static
		input_format : RGB888_U8
		csc_switch : true
		rbuv_swap_switch : false
		matrix_r0c0 : 47
		matrix_r0c1 : 157
		matrix_r0c2 : 16
		matrix_r1c0 : -26
		matrix_r1c1 : -87
		matrix_r1c2 : 112
		matrix_r2c0 : 112
		matrix_r2c1 : -102
		matrix_r2c2 : -10
		output_bias_0 : 16
		output_bias_1 : 128
		output_bias_2 : 128
		}

RGB888_U8 to YVU444SP_U8

Table 3-6 RGB888_U8 to YVU444SP_U8

JPEG	BT-601NARROW	BT-709NARROW

aipp_op {
aipp_mode: static
input_format : RGB888_U8
csc_switch : true
rbuv_swap_switch : false
matrix_r0c0 : 77
matrix_r0c1 : 150
matrix_r0c2 : 29
matrix_r1c0 : 128
matrix_r1c1 : -107
matrix_r1c2 : -21
matrix_r2c0 : -43
matrix_r2c1 : -85
matrix_r2c2 : 128
output_bias_0 : 0
output_bias_1 : 128
output_bias_2 : 128
}
	aipp_op {
	aipp_mode: static
	input_format : RGB888_U8
	csc_switch : true
	rbuv_swap_switch : false
	matrix_r0c0 : 66
	matrix_r0c1 : 129
	matrix_r0c2 : 25
	matrix_r1c0 : 112
	matrix_r1c1 : -94
	matrix_r1c2 : -18
	matrix_r2c0 : -38
	matrix_r2c1 : -74
	matrix_r2c2 : 112
	output_bias_0 : 16
	output_bias_1 : 128
	output_bias_2 : 128
	}
		aipp_op {
		aipp_mode: static
		input_format : RGB888_U8
		csc_switch : true
		rbuv_swap_switch : false
		matrix_r0c0 : 47
		matrix_r0c1 : 157
		matrix_r0c2 : 16
		matrix_r1c0 : 112
		matrix_r1c1 : -102
		matrix_r1c2 : -10
		matrix_r2c0 : -26
		matrix_r2c1 : -87
		matrix_r2c2 : 112
		output_bias_0 : 16
		output_bias_1 : 128
		output_bias_2 : 128
		}

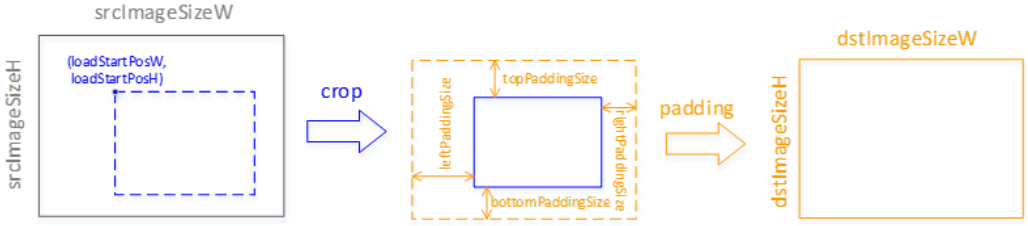
RGB888_U8 to Gray

```
aipp_op {
   aipp_mode: static
   input_format : RGB888_U8
   csc_switch : true
   rbuv_swap_switch : false
   matrix_r0c0 : 76
   matrix_r0c1 : 150
   matrix_r0c2 : 30
   matrix_r1c0 : 0
   matrix_r1c1 : 0
   matrix_r1c2 : 0
   matrix_r2c0 : 0
   matrix_r2c1 : 0
   matrix_r2c2 : 0
   output_bias_0 : 0
   output_bias_1 : 0
   output_bias_2 : 0
}
```

3.4 Crop/Padding Configuration

Currently, AIPP supports image resize by crop and padding.

Figure 3-1 Image resize



For YUV420SP_U8 images, the values of **load_start_pos_w**, **load_start_pos_h**, **crop_size_w**, and **crop_size_h4** must be even numbers. The width/height of the cut out image must be the same as **W**/ **H** defined in the network input.

A configuration example is as follows:

```
aipp_op {
   aipp_mode: static
   input_format : YUV400_U8

   src_image_size_w :320
   src_image_size_h :240

   crop :true
   load_start_pos_w :10
   load_start_pos_h :20
   crop_size_w :50
   crop_size_h :60

   padding : true
   left_padding_size :10
   right_padding_size :20
   top_padding_size :10
   bottom_padding_size :20
}
```

3.5 AIPP Configurations According to Model Inputs

You can specify more than one set of AIPP configurations in the AIPP configuration file. Different AIPP processing is performed for different model inputs. Different sets of AIPP parameters are separated by the **aipp_op** flags. For example:

```
aipp_op {
      related_input_rank: 0
      aipp_mode : dynamic
      max_src_image_size: 60000
}
aipp_op {
      related_input_rank : 1
      aipp_mode : dynamic
      max_src_image_size: 80000
}
```

The preceding configuration defines two sets of AIPP parameters, corresponding to AIPP on the first and second inputs of the model. Dynamic AIPP is adopted. The maximum image sizes allowed by the two inputs are 60000 bytes and 80000 bytes respectively.

3.6 AIPP Verification of the Model Input Size

If static or dynamic AIPP is configured, the size (**input_size**) of the generated image received by the Da Vinci model is changed due to operations such as crop and padding. In the model inference phase, the OME verifies the size of the input image. In dynamic AIPP mode, the input size must be less than or equal to **max_src_image_size**. In static AIPP mode, calculate the input size according to **Table 3-7**.

For static AIPP, assume that the batch count of the model is **N**, the image width is **src_image_size_w**, and the height width is **src_image_size_h**. Table 3-7 describes the verification formulas for the size of the model input.

Table 3-7 input_size verification formulas

input_format	input_size
YUV400_U8	N * src_image_size_w * src_image_size_h
YUV420SP_U8	N * src_image_size_w * src_image_size_h * 1.5
XRGB8888_U8	N * src_image_size_w * src_image_size_h * 4
RGB888_U8	N * src_image_size_w * src_image_size_h * 3

The OMG adds a model input before a model with dynamic AIPP is generated, according to the following formula:

sizeof(kAippDynamicPara) + (batch_count – 1) x sizeof(kAippDynamicBatchPara)

3.7 Parameter Structure for Dynamic AIPP

```
typedef struct tagAippDynamicBatchPara
{
    int8_t cropSwitch;              //crop switch
    int8_t scfSwitch;               //resize switch
    int8_t paddingSwitch;   // 0: unable padding,
                            // 1: padding config value,sfr_filling_hblank_ch0 ~     sfr_filling_hblank_ch2
                            // 2: padding source picture data, single row/collumn copy
                            // 3: padding source picture data, block copy
                            // 4: padding source picture data, mirror copy
int8_t rotateSwitch; //Rotation switch: 0 indicates zero rotation, 1 indicates 90° clockwise rotation, 2
indicates 180° clockwise rotation, and 3 indicates 270° clockwise rotation.
    int8_t reserve[4];
    int32_t cropStartPosW;          //the start horizontal position of cropping
    int32_t cropStartPosH;          //the start vertical position of cropping
    int32_t cropSizeW;              //crop width
    int32_t cropSizeH;              //crop height
    int32_t scfInputSizeW;          //input width of scf
    int32_t scfInputSizeH;          //input height of scf
    int32_t scfOutputSizeW;         //output width of scf
    int32_t scfOutputSizeH;         //output height of scf
    int32_t paddingSizeTop;         //top padding size
    int32_t paddingSizeBottom;      //bottom padding size
    int32_t paddingSizeLeft;        //left padding size
    int32_t paddingSizeRight;       //right padding size
    int16_t dtcPixelMeanChn0;       //mean value of channel 0
    int16_t dtcPixelMeanChn1;       //mean value of channel 1
    int16_t dtcPixelMeanChn2;       //mean value of channel 2
    int16_t dtcPixelMeanChn3;       //mean value of channel 3
#ifndef DAVINCI_TINY
    uint16_t dtcPixelMinChn0;       //min value of channel 0
    uint16_t dtcPixelMinChn1;       //min value of channel 1
    uint16_t dtcPixelMinChn2;       //min value of channel 2
    uint16_t dtcPixelMinChn3;       //min value of channel 3
    uint16_t dtcPixelVarReciChn0;   //sfr_dtc_pixel_variance_reci_ch0
    uint16_t dtcPixelVarReciChn1;   //sfr_dtc_pixel_variance_reci_ch1
    uint16_t dtcPixelVarReciChn2;   //sfr_dtc_pixel_variance_reci_ch2
    uint16_t dtcPixelVarReciChn3;   //sfr_dtc_pixel_variance_reci_ch3
```



```
    int8_t reserve1[16];            //32B assign, for ub copy
#endif
}kAippDynamicBatchPara;
typedef struct tagAippDynamicPara
{
uint8_t inputFormat; //Input format: YUV420SP_U8, XRGB8888_U8, or RGB888_U8
    //uint8_t outDataType; //output data type: CC_DATA_HALF,CC_DATA_INT8, CC_DATA_UINT8
    int8_t cscSwitch;               //csc switch
    int8_t rbuvSwapSwitch;          //rb/ub swap switch
    int8_t axSwapSwitch;            //RGBA->ARGB, YUVA->AYUV swap switch
    int8_t batchNum;                //batch parameter number
    int8_t reserve1[3];
    int32_t srcImageSizeW;          //source image width
    int32_t srcImageSizeH;          //source image height
    int16_t cscMatrixR0C0;          //csc_matrix_r0_c0
    int16_t cscMatrixR0C1;          //csc_matrix_r0_c1
    int16_t cscMatrixR0C2;          //csc_matrix_r0_c2
    int16_t cscMatrixR1C0;          //csc_matrix_r1_c0
    int16_t cscMatrixR1C1;          //csc_matrix_r1_c1
    int16_t cscMatrixR1C2;          //csc_matrix_r1_c2
    int16_t cscMatrixR2C0;          //csc_matrix_r2_c0
    int16_t cscMatrixR2C1;          //csc_matrix_r2_c1
    int16_t cscMatrixR2C2;          //csc_matrix_r2_c2
    int16_t reserve2[3];
    uint8_t cscOutputBiasR0;   //output bias for RGB to YUV, element of row 0, unsigned number
    uint8_t cscOutputBiasR1;   //output bias for RGB to YUV, element of row 1, unsigned number
    uint8_t cscOutputBiasR2;   //output bias for RGB to YUV, element of row 2, unsigned number
    uint8_t cscInputBiasR0;    //input bias for YUV to RGB, element of row 0, unsigned number
    uint8_t cscInputBiasR1;    //input bias for YUV to RGB, element of row 1, unsigned number
    uint8_t cscInputBiasR2;    //input bias for YUV to RGB, element of row 2, unsigned number
    uint8_t reserve3[2];
    int8_t reserve4[16];            //32B assign, for ub copy
    kAippDynamicBatchPara aippBatchPara;  //allow transfer several batch para.
} kAippDynamicPara;
```


4 Quantization Configuration

4.1 Overview

Quantization herein refers to low-bit quantization performed on high-accuracy data, so as to save the network memory usage, reduce the transmission delay, and improve the operation efficiency.

Currently, quantization using the Convolution, Full Connection, and ConvolutionDepthwise operators is supported, including weight, offset, and data quantization. Two quantization modes are supported: non-offset mode and data-offset mode.

The quantized weight and offset are stored in the Da Vinci model during Da Vinci model generation. In model inference, the quantized weight and offset are used for data computation.

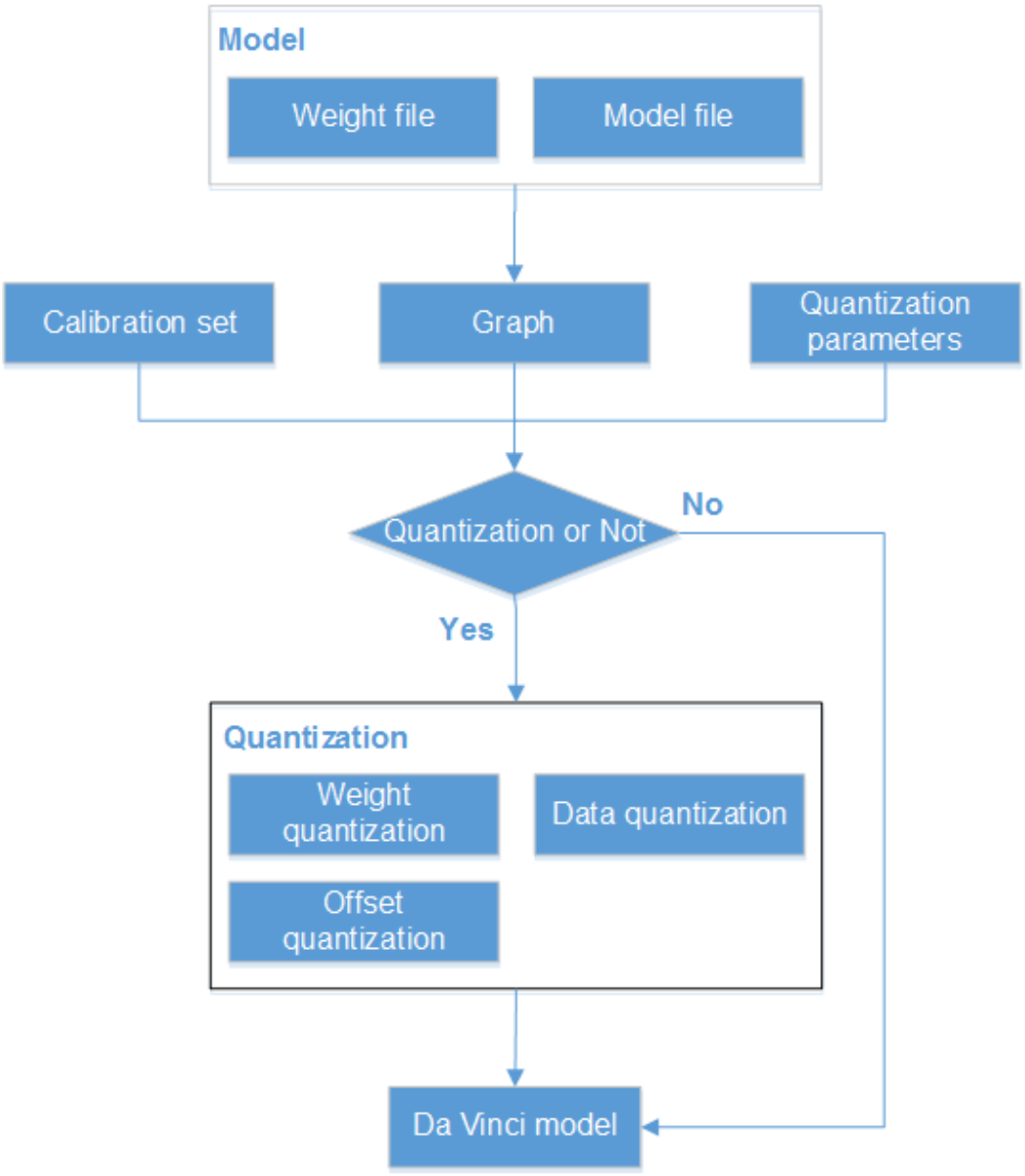
The concepts are described as follows:

- Non-offset mode: Both the weight and data are offset.
- Data-offset mode: The data is offset while the weight is not.
- Weight quantification: Performs int8 quantization on the weight file according to the quantization algorithm. In with-offset quantization mode, the int8 weight, scale, and offset are output. In non-offset quantization mode, the int8 weight and scale are output. Currently, only the non-offset quantization mode is supported for weight quantization.
- Data quantification: Performs inference by using limited inputs (a calibration set, used to train quantization parameters and ensure the accuracy). The scale and offset of the quantized data are obtained in the data-offset quantization mode or the scale of the quantized data is obtained in non-offset

quantization mode, according to the frequency statistics and divergence algorithm.

- Offset quantization: Quantizes the FP32 offset data into the INT32 output based on the weight-quantized scale and data-quantized scale

Figure 4-1 Quantization processing process



4.2 Configuration File Template

Quantization is supported in single-input and multi-input network scenarios. You can use this configuration file template directly or make modifications as required for adaptation. Note that some of the parameters in the configuration file are mandatory. **4.3 Description of the Configuration Parameters** describes the parameters in the configuration file and whether a parameter is mandatory.

The configuration file cannot contain Chinese characters.

Single-Input Network Scenario

With the image input, the **omg** command specifies the quantization configuration. The following is an example of the configuration file:

```
device:USE_CPU
quantize_algo:HALF_OFFSET
weight_type:VECTOR_TYPE
bin:150
type:KL
inference_with_data_quantized:false
inference_with_weight_quantized:true
super_parameter:
{
min_percentile:PERCENTILE_HIGH
max_percentile:PERCENTILE_MID
start_ratio:0.7
end_ratio:1.3
step_ratio:0.01
}
exclude_op:'fc1000'
batch_count:50
preprocess_parameter:
{
input_type:IMAGE
image_format:BGR
input_file_path:'calibration/image_set'
mean_value:104.0
mean_value:117.0
mean_value:123.0
standard_deviation:1.0
}
```

Multi-Input Network Scenario

With image+binary inputs, the **omg** command specifies the quantization configuration. The following is an example of the configuration file:

```
device:USE_CPU
quantize_algo:HALF_OFFSET
weight_type:VECTOR_TYPE
bin:150
type:KL
inference_with_data_quantized:false
inference_with_weight_quantized:true
super_parameter:
{
min_percentile:PERCENTILE_HIGH
max_percentile:PERCENTILE_MID
start_ratio:0.7
end_ratio:1.3
step_ratio:0.01
}
exclude_op:'fc1000'
batch_count:50
preprocess_parameter:
{
input_type:IMAGE
image_format:BGR
input_file_path:'calibration/image_set'
mean_value:104.0
mean_value:117.0
mean_value:123.0
standard_deviation:1.0
}
preprocess_parameter:
{
input_type:BINARY
```



```
input_file_path:'calibration/img_info.bin'
}
```

4.3 Description of the Configuration Parameters

Table 4-1 Description of the device parameter

Function	Inference mode
Type	enum
Value Range	USE_CPU
Parameter	**USE_CPU**: Uses the CPU for inference.
Recommended Value	USE_CPU
Mandatory or Optional	Mandatory

Table 4-2 Description of the quantize_algo parameter

Function	Quantization mode
Type	enum
Value Range	• NON_OFFSET • HALF_OFFSET
Parameter	• **NON_OFFSET**: Both the weight and data are offset. • **HALF_OFFSET**: The data is offset while the weight is not. There are two quantization mapping formulas: • With offset: q_uint8 = round (d_float/scale) – offset • Without offset: q_int8 = round (d_float/scale)
Recommended Value	HALF_OFFSET
Mandatory or Optional	Mandatory

Table 4-3 Description of the weight_type parameter

Function	Weight quantization mode
Type	enum

Value Range	• VECTOR_TYPE • SCALAR_TYPE
Parameter	A convolution operator may have multiple filters, and the corresponding quantization parameters may be different. • **VECTOR_TYPE**: Each filter uses a separate group of quantization parameters • **SCALAR_TYPE**: The filters share the same group of quantization parameters.
Recommended Value	VECTOR_TYPE
Mandatory or Optional	Mandatory

Table 4-4 Description of the preprocess_parameter parameters

Function	Pre-processing and input parameters
Type	Struct
Value Range	None
Parameter	Pre-processing and input settings. The number of **preprocess_parameter** parameters must be the same as the number of operators input to the network. **preprocess_parameter** contains the following parameters: • input_type • image_format • input_file_path • mean_value • standard_deviation
Recommended Value	None
Mandatory or Optional	Mandatory

Table 4-5 Description of the input_type parameter

Function	Input data type
Type	enum
Value Range	• IMAGE • BINARY

Parameter	Input type. Currently, the image (**IMAGE**) and binary file (**BINARY**) formats are supported. • **IMAGE**: image format The following image formats are supported: .bmp, .dib, .jpeg, .jpg, .jpe, .jp2, .png, .webp, .pbm, .pgm, .ppm, .sr, .ras, .tiff, .tif, .BMP, .DIB, .JPEG, .JPG, .JPE, .JP2, .PNG, .WEBP, .PBM, .PGM, .PPM, .SR, .RAS, .TIFF, .TIF • **BINARY**: binary file format. For details, see **Table 4-6**. For non-4D (num, channels, height, and width) data, the data needs to be padded to four dimensions and the value of a padded dimension is **1**.
Recommended Value	None
Mandatory or Optional	Mandatory

Table 4-6 Binary file formats

File Header/ Data	Address Offset	Type	Value	Description
File header (20 bytes in total)	0000	32-bit int	510	Magic number. When the magic number is **510**, it is used to verify the validity of a file.
	0004	32-bit int	50	Input num
	0008	32-bit int	3	Input channels
	0012	32-bit int	28	Input height
	0016	32-bit int	28	Input width
Data	...	float	126	The number of data records is equal to the product of (num x channels x height x width).

Table 4-7 Description of the input_file_path parameter

Function	Input data address, that is, path of the calibration set images
Type	string

Value Range	None
Parameter	Folder or file to be quantified. The path cannot contain Chinese characters, special characters (including **| ; & $ > <** `), or spaces. Set this parameter based on the actual network configuration. ● For **IMAGE** input, specify the folder. ● For **BINARY** input, specify the file.
Recommended Value	Set this parameter to an image folder or a binary file as required.
Mandatory or Optional	Mandatory

Table 4-8 Description of the image_format parameter

Function	Order of the three channels of the image input
Type	enum
Value Range	● BGR ● RGB
Parameter	Order of the three channels of the image input for model training This parameter is mandatory when **input_type** is set to **IMAGE**. This parameter is determined based on the order of the input channels for network training. A typical order is BGR.
Recommended Value	BGR
Mandatory or Optional	Mandatory

Table 4-9 Description of the mean_value parameter

Function	Mean value for image preprocessing
Type	float
Value Range	[0, 255.0]

Parameter	Mean value of a channel for image preprocessing This parameter is mandatory when **input_type** is set to **IMAGE**. The count of mean values is determined by the channel count. Three mean values need to be configured for the RGB channels respectively as follows: • mean_value: 104.0 • mean_value: 117.0 • mean_value: 123.0
Recommended Value	None
Mandatory or Optional	Mandatory

Table 4-10 Description of the standard_deviation parameter

Function	Standard deviation for image preprocessing
Type	float
Value Range	[0, FLOAT_MAX]
Parameter	Standard deviation for image preprocessing This parameter is mandatory when **input_type** is set to **IMAGE**. Channels share the same standard deviation. If the input range is beyond the range that can be represented by the FLOAT type, the quantization precision of the model cannot be guaranteed. If 0 ≤ standard_deviation ≤ 0.00001, the value of **standard_deviation** is **1.0**.
Recommended Value	**1.0**: The image value range (Source image value – Mean value) does not change. **255.0**: If the image value range is [0, 0,255], the value can be condensed to [0,1, 1.0].
Mandatory or Optional	Mandatory

Table 4-11 Description of the bin parameter

Function	Data range of the mapping histogram
Type	uint32
Value Range	[0, 1000]

Parameter	Data range of the histogram statistics The statistics histogram is required during the divergence computation. The value of this parameter determines the maximum value of the histogram. If this parameter is not set or set to **0**, the default value **150** is used.
Recommended Value	100/150/200/250
Mandatory or Optional	Optional

Table 4-12 Description of the type parameter

Function	Type of the divergence computation
Type	enum
Value Range	• KL • SYMKL • JSD
Parameter	• **KL**: Kullback-Leibler Divergence • **SYMKL**: Symmetric Kullback-Leibler Divergence • **JSD**: Jensen-Shannon Divergence Different divergence types correspond to different computation methods. The default value is **KL**.
Recommended Value	None
Mandatory or Optional	Optional

Table 4-13 Description of the inference_with_data_quantized parameter

Function	Whether to use the quantized data as the inference input
Type	bool
Value Range	• true • false
Parameter	Whether to use the quantized data as the inference input When this parameter is set to **true**, the quantization of the input data is simulated. The default value is **false**.
Recommended Value	false

Mandatory or Optional	Optional

Table 4-14 Description of the inference_with_weight_quantized parameter

Function	Whether to use the quantized weight as the inference input
Type	bool
Value Range	• true • false
Parameter	Whether to use the quantized weight as the inference input When this parameter is set to **true**, the quantization and de-quantization of the weight data are simulated. The default value is **true**.
Recommended Value	true
Mandatory or Optional	Optional

Table 4-15 Description of the super_parameter parameters

Function	Search parameters
Type	Struct
Value Range	None
Parameter	Search parameters **super_parameter** contains the following parameters: • min_percentile • max_percentile • start_ratio • end_ratio • step_ratio
Recommended Value	None
Mandatory or Optional	Optional

Table 4-16 Description of the min_percentile parameter

Function	Searched minimum number
Type	enum
Value Range	• PERCENTILE_HIGH • PERCENTILE_MID • PERCENTILE_LOW
Parameter	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. • PERCENTILE_HIGH: 1.0 • PERCENTILE_MID: 0.99999 • PERCENTILE_LOW: 0.9999
Recommended Value	PERCENTILE_HIGH
Mandatory or Optional	Optional

Table 4-17 Description of the max_percentile parameter

Function	Searched maximum number
Type	enum
Value Range	• PERCENTILE_HIGH • PERCENTILE_MID • PERCENTILE_LOW
Parameter	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. • PERCENTILE_HIGH: 1.0 • PERCENTILE_MID: 0.99999 • PERCENTILE_LOW: 0.9999
Recommended Value	PERCENTILE_MID/PERCENTILE_HIGH
Mandatory or Optional	Optional

Table 4-18 Description of the start_ratio, end_ratio, and step_ratio parameters

Function	Parameter
Type	float
Value Range	end_ratio > start_ratio > 0, step_ratio > 0 (If the input range is beyond the range that can be represented by the FLOAT type, the quantization accuracy of the model cannot be guaranteed.)
Parameter	• **start_ratio** indicates the search start. • **end_ratio** indicates the search end. • **step_ratio** indicates the search step. After **d_max** and **d_min** are found in the algorithm, the ranges taken before and after **d_max** and **d_min** are determined based on **start_ratio** and **end_ratio**. Then, the step incremented each time is determined by **step_ratio**. For example: If **d_max** is **100**, **start_ratio** is **0.8**, **end_ratio** is **1.2**, and **step_ratio** is **0.01**, the defined search range boundaries before and after **d_max** are **80** (100 x 0.8) and **120** (100 x 1.2) at a step of **1** (100 x 0.01). That is, there are 41 **d_max** search results in total.
Recommended Value	The following two sets of configurations are recommended: • start_ratio: 0.7 end_ratio: 1.3 step_ratio: 0.01 • start_ratio: 0.3 end_ratio: 1.7 step_ratio: 0.01
Mandatory or Optional	Optional

Table 4-19 Description of the batch_count parameter

Function	Number of images in the quantization calibration set to be processed
Type	uint32
Value Range	[0, UINT32_MAX)
Parameter	Number of images in the quantization calibration set to be processed If this parameter is not set or set to **0**, all images in the calibration set are used. If the value is greater than **0**, the smaller one in the total number of images in the calibration set path and the value of this parameter is used as the actual number of images in the calibration set. It is recommended that the number of images in the calibration set do not exceed **500**.

Recommended Value	50
Mandatory or Optional	Optional

Table 4-20 Description of the exclude_op parameter

Function	Quantization operator blacklist
Type	string
Value Range	Operator name
Parameter	Operators in the list are not used for quantization. • Only the Convolution, Full Connection, and ConvolutionDepthwise operators are supported. • To blacklist multiple operators, multiple **exclude_op** parameters need to be set. Each parameter occupies a line. For example: **exclude_op:'aaa'** **exclude_op:'bbb'**
Recommended Value	None
Mandatory or Optional	Optional

4.4 Parameter Tuning

The weight, offset, and data are quantized based on the quantization parameters set during model conversion. If the accuracy does not meet requirements, you can perform the following steps to adjust the quantization parameters.

1. Calibrate the model data and parameters based on the default parameter values described in **4.2 Configuration File Template**.
2. If the quantization accuracy obtained in Step **1** is as expected, the parameter tuning ends. Otherwise, go to Step **3**.
3. Manually perform the following modifications and then perform quantization.
   a. Modify the search range parameters.

Table 4-21 Search scope parameters

As It Is	To Be
start_ratio: 0.7 end_ratio: 1.3 step_ratio: 0.01	start_ratio: 0.3 end_ratio: 1.7 step_ratio: 0.01

b. Change the value of **bin** from **150** to **200** or **250**.

c. For a detection network, you are advised to set **max_percentile** to **PERCENTILE_MID** (indicating **0.99999**).

4. If the quantization accuracy obtained in Step **3** is as expected, the parameter tuning ends. Otherwise, go to Step **5**.

5. Traverse the configuration options of the quantization parameters described in **Table 4-22**. (You can adjust the search range based on the calibration time requirement.)

Table 4-22 Parameter Settings

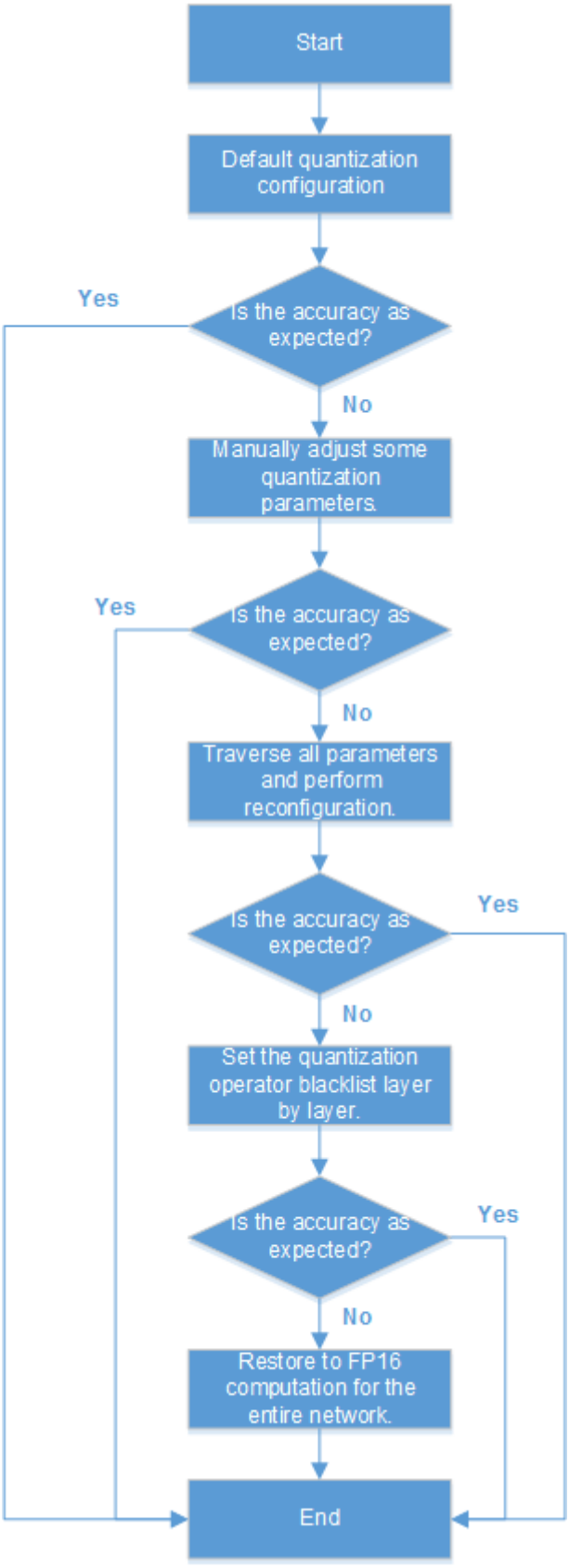
Parameter	Configuration Description
quantize_algo	**HALF_OFFSET**: The data is offset while the weight is not.
weight_type	**VECTOR_TYPE**: Each filter uses a separate group of quantization parameters.
bin	The statistics histogram is required during the divergence computation. The value of this parameter determines the maximum value of the histogram. If this parameter is not set or set to **0**, the default value **150** is used. **100**, **150**, and **200** are recommended.
type	Different divergence types correspond to different computation methods. The default value is **KL**. ● **KL**: Kullback-Leibler Divergence ● **SYMKL**: Symmetric Kullback-Leibler Divergence ● **JSD**: Jensen-Shannon Divergence
● start_ratio ● end_ratio ● step_ratio	**start_ratio** indicates the search start. **end_ratio** indicates the search end. **step_ratio** indicates the search step. The following two sets of configurations are recommended: ● start_ratio: 0.7 end_ratio: 1.3 step_ratio: 0.01 ● start_ratio: 0.3 end_ratio: 1.7 step_ratio: 0.01
max_percentile	Maximum number to be considered as the search result among a group of numbers in descending order **1.0** and **0.99999** are recommended.

Parameter	Configuration Description
batch_count	Number of images in the quantization calibration set to be processed The value can be an integer greater than 1. **10**, **20**, and **50** are recommended.

6. If the quantization accuracy obtained in Step **5** is as expected, the parameter tuning ends. Otherwise, go to Step **7**.
7. Use the **exclude_op** parameter to set the quantization operator blacklist. Operators are blacklisted layer by layer from the first layer. Operators in the operator blacklist are not quantized, narrowing down the operators that affect the accuracy.

   Only the operators in the blacklist are not quantized. In this case, int8 computation and FP16 computation are not separated.
8. If the quantization accuracy obtained in Step **7** is as expected, the parameter tuning ends. Otherwise, it indicates that the quantization does not affect the accuracy, and the quantization configuration is not required. Remove the quantization configuration, and restore the network-wide FP16 computation.

Figure 4-2 Parameter tuning process



5 FAQs

5.1 What Do I Do If No Log Is Output to the Screen When the DDK Installation User Is HwHiAiUser?

In the scenario where the DDK is installed by the **HwHiAiUser** user, the log is stored in the **/var/dlog** directory on the host by default. You can also set the following environment variable to enable log output to the screen:

```
export  SLOG_PRINT_TO_STDOUT=1
```

If the DDK installation user is not **HwHiAiUser**, the log is output to the screen by default.

5.2 What Do I Do If Model Conversion Takes Too Long When the OS and Architecture Configuration of the DDK Server is Arm (aarch64)?

For an Arm (AArch64) DDK server, to accelerate model conversion, you can use the numactl tool to specify CPU cores for model conversion as follows:

1. Log in to the DDK server as the DDK installation user and run the **su root** command to switch to the **root** user.
2. Ensure that the host server is connected to the network, and run the following command to install the numactl tool:

   ```
   yum -y install numactl
   ```


3. Switch to the DDK installation user and run the **numactl -C** command to specify CPU cores 16–31 to process model conversion:

   CPU cores 16–31 are recommended for better processing performance. You can also change the CPU cores based as required.

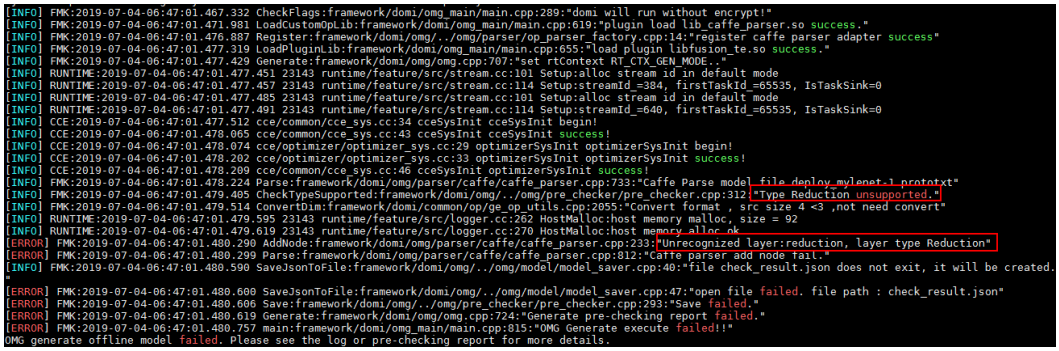
   ```
   numactl -C 16-31 --localalloc omg <args>
   ```

5.3 What Do I Do If the Error "Unrecognized layer:xxx, layer type xxx" Is Reported During Model Conversion?

Symptom

When the omg command is run to convert a model, the error messages "Type XXX unsupported" and "Unrecognized layer:xxx, layer type XXX" are displayed, as shown in **Figure 5-1**.

Figure 5-1 Model conversion error messages



NOTE

As shown in **Figure 5-1**, the Reduction operator is used as an example for illustration.

Solution

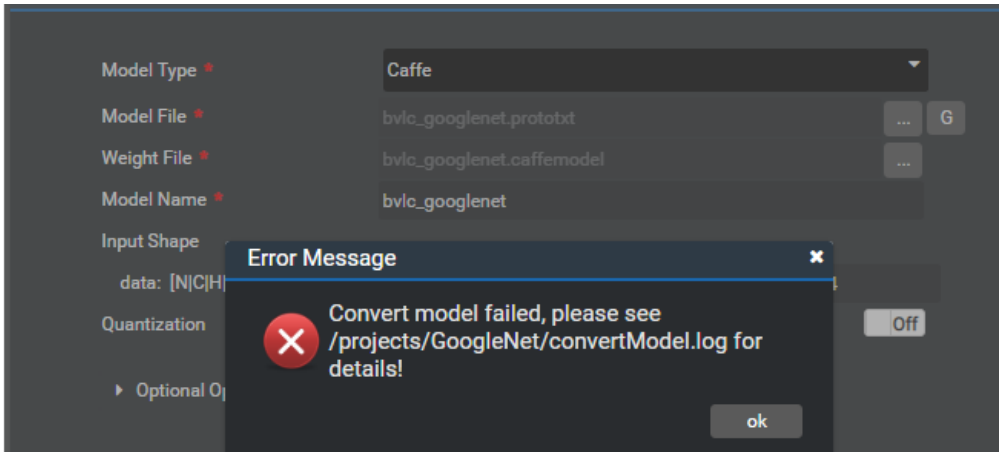
It is because the source model contains operators that cannot be identified by Ascend 310. You need to develop these operators, develop operator plug-ins, and load operator plug-ins to convert the model, according to *TE Custom Operator Development Guide*.

5.4 What Do I Do If the Error "It is recommended to convert layers-structure to layer-structure by caffe tool" Is Reported During Model Conversion?

Symptom

The .prototxt and .caffemodel files of GoogLeNet downloaded from https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet fail to be imported, as shown in **Figure 5-2**.

Figure 5-2 Model conversion error message



The **convertModel.log** file shows the following error information.

```
[ERROR] FMK:2018-12-22-23:47:48.147.318 ConvertNetParameter:framework/domi/omg/parser/caffe/caffe_parser.cpp:776:"The weight file is consisted of layers-structure which is deprecated in caffe and unsupport in OMG. It is recommended to convert layers-structure to layer-structure by caffe tool. Error Code:0xFFFFFFFF(failed)"
```

Solution

It is because the .prototxt and .caffemodel files have to be upgraded to the latest by using tools provided by Caffe. You can obtain the Caffe tools from **Link**.

Step 1 Download the upgrade_net_proto_text and upgrade_net_proto_binary tools to any directory on the Mind Studio server.

Step 2 Compile and obtain tools upgrade_net_proto_text and upgrade_net_proto_binary by referring to **Link**.

Step 3 Use the upgrade_net_proto_text tool to upgrade the .prototxt file.

upgrade_net_proto_text *model_old.prototxt model_new.prototxt*

Step 4 Use the upgrade_net_proto_binary tool to upgrade the .caffemodel file.

upgrade_net_proto_binary *model_old.caffemodel model.new.caffemodel*

Step 5 Use the upgraded .protoxt file and .caffemodel file to import the model again.

----End

6 Appendix

6.1 Change History

Release Date	Description
2020-05-30	This issue is the first official release.

