1 Operation Guide...............................................................................................................................................................1
2 Data Management...............................................................................................................................................................5
  2.1 Data Management Overview........................................................................................................................................5
  2.2 Creating a Dataset..........................................................................................................................................................9
  2.3 Labeling Data...............................................................................................................................................................21
    2.3.1 Image Classification...............................................................................................................................................21
    2.3.2 Object Detection....................................................................................................................................................27
    2.3.3 Text Classification...............................................................................................................................................33
    2.3.4 Named Entity Recognition..................................................................................................................................36
    2.3.5 Text Triplet.............................................................................................................................................................39
    2.3.6 Sound Classification..............................................................................................................................................43
    2.3.7 Speech Labeling.....................................................................................................................................................46
    2.3.8 Speech Paragraph Labeling..................................................................................................................................48
    2.3.9 Video Labeling.......................................................................................................................................................50
  2.4 Importing Data...............................................................................................................................................................52
    2.4.1 Import Operation...................................................................................................................................................52
    2.4.2 Specifications for Importing Data from an OBS Directory....................................................................................55
    2.4.3 Specifications for Importing the Manifest File......................................................................................................59
  2.5 Exporting Data...............................................................................................................................................................70
  2.6 Modifying a Dataset.......................................................................................................................................................73
  2.7 Publishing a Dataset.....................................................................................................................................................74
  2.8 Deleting a Dataset.........................................................................................................................................................76
  2.9 Managing Dataset Versions...........................................................................................................................................76
  2.10 Auto Labeling..............................................................................................................................................................78
  2.11 Confirming Hard Examples.........................................................................................................................................81
  2.12 Auto Grouping.............................................................................................................................................................85
  2.13 Data Features..............................................................................................................................................................87
  2.14 Deploying a Model in One Click..................................................................................................................................93
  2.15 Team Labeling............................................................................................................................................................95
    2.15.1 Team Labeling Overview..................................................................................................................................95
    2.15.2 Team Management..............................................................................................................................................97
    2.15.3 Member Management...........................................................................................................................................98
    2.15.4 Managing Team Labeling Tasks...........................................................................................................................100
### 3 DevEnviron (Notebook)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Introduction to Notebook</td>
<td>107</td>
</tr>
<tr>
<td>3.2 Managing Notebook Instances</td>
<td>109</td>
</tr>
<tr>
<td>3.2.1 Creating a Notebook Instance</td>
<td>109</td>
</tr>
<tr>
<td>3.2.2 Creating a Notebook Instance with a Git Repository</td>
<td>113</td>
</tr>
<tr>
<td>3.2.3 Opening a Notebook Instance</td>
<td>118</td>
</tr>
<tr>
<td>3.2.4 Starting or Stopping a Notebook Instance</td>
<td>119</td>
</tr>
<tr>
<td>3.2.5 Deleting a Notebook Instance</td>
<td>119</td>
</tr>
<tr>
<td>3.3 Using Jupyter Notebook</td>
<td>120</td>
</tr>
<tr>
<td>3.3.1 Introduction to Jupyter Notebook</td>
<td>120</td>
</tr>
<tr>
<td>3.3.2 Common Operations on Jupyter Notebook</td>
<td>120</td>
</tr>
<tr>
<td>3.3.3 Configuring the Jupyter Notebook Environment</td>
<td>123</td>
</tr>
<tr>
<td>3.3.3.1 Using the Notebook Terminal Function</td>
<td>123</td>
</tr>
<tr>
<td>3.3.3.2 Switching the CUDA Version on the Terminal Page of a GPU-based Notebook Instance</td>
<td>124</td>
</tr>
<tr>
<td>3.3.3.3 Installing External Libraries and Kernels in Notebook Instances</td>
<td>125</td>
</tr>
<tr>
<td>3.3.4 Uploading and Downloading Data</td>
<td>127</td>
</tr>
<tr>
<td>3.3.4.1 Uploading Large Files to a Notebook Instance</td>
<td>127</td>
</tr>
<tr>
<td>3.3.4.2 Downloading Large Files in Notebook Instances to a Local PC</td>
<td>128</td>
</tr>
<tr>
<td>3.3.5 Using ModelArts SDKs</td>
<td>129</td>
</tr>
<tr>
<td>3.3.6 Synchronizing Files with OBS</td>
<td>130</td>
</tr>
<tr>
<td>3.3.7 Using the Convert to Python File Function</td>
<td>131</td>
</tr>
<tr>
<td>3.4 Using JupyterLab</td>
<td>131</td>
</tr>
<tr>
<td>3.4.1 JupyterLab Overview and Common Operations</td>
<td>132</td>
</tr>
<tr>
<td>3.4.2 Uploading and Downloading Data</td>
<td>137</td>
</tr>
<tr>
<td>3.4.2.1 Uploading Data to JupyterLab</td>
<td>137</td>
</tr>
<tr>
<td>3.4.2.2 Downloading a File from JupyterLab</td>
<td>138</td>
</tr>
<tr>
<td>3.4.3 Using Sample Notebook Instances of ModelArts</td>
<td>139</td>
</tr>
<tr>
<td>3.4.4 Using ModelArts SDKs</td>
<td>140</td>
</tr>
<tr>
<td>3.4.5 Synchronizing Files with OBS</td>
<td>141</td>
</tr>
<tr>
<td>3.4.6 Using the Git Plug-in</td>
<td>142</td>
</tr>
<tr>
<td>3.4.7 Setting the Initialization Script</td>
<td>147</td>
</tr>
</tbody>
</table>

### 4 Algorithm Management

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Overview</td>
<td>151</td>
</tr>
<tr>
<td>4.2 Creating an Algorithm</td>
<td>151</td>
</tr>
<tr>
<td>4.3 Creating a Training Job Using an Algorithm</td>
<td>157</td>
</tr>
</tbody>
</table>

### 5 Training Management

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Model Training Overview</td>
<td>159</td>
</tr>
<tr>
<td>5.2 Built-in Algorithms</td>
<td>160</td>
</tr>
<tr>
<td>5.2.1 Introduction to Built-in Algorithms</td>
<td>160</td>
</tr>
<tr>
<td>5.2.2 Requirements on Datasets</td>
<td>161</td>
</tr>
<tr>
<td>5.2.3 Algorithms and Their Running Parameters</td>
<td>163</td>
</tr>
<tr>
<td>5.3 Creating a Training Job</td>
<td>177</td>
</tr>
</tbody>
</table>
6.4.8.3 Analysis on Background Errors of Object Detection Models and Solution...........................................298
6.4.8.4 Analysis on the Sensitivity of Object Detection Models to Bounding Box Aspect Ratios and Solution..........................................................................................................................302
6.4.8.5 Analysis on the Sensitivity of Object Detection Models to Bounding Box Brightness and Solution..................................................................................................................................................................................302
6.4.8.6 Analysis on the Sensitivity of Object Detection Models to Bounding Box Clarity and Solution...............308
6.4.8.7 Analysis on the Sensitivity of Object Detection Models to Bounding Box Areas and Solution..............312
6.4.8.8 Analysis on the Sensitivity of Object Detection Models to Bounding Box Marginalization Degrees and Solution.........................................................................................................................................................................................315
6.4.8.9 Analysis on the Sensitivity of Object Detection Models to Bounding Box Overlap Degrees and Solution.........................................................................................................................................................................................................319
6.4.8.10 Fine-grained Classification Optimization Using Center Loss.................................................................322
6.4.8.11 Common Methods of Optimizing Model Precision in Model Optimization...........................................326
6.4.8.12 Data Augmentation Based on Model Evaluation..........................................................................................331
6.4.8.13 Model Attack Based on Sample Preprocessing.........................................................................................335

7 Model Deployment..................................................................................................................................................338

7.1 Model Deployment Overview........................................................................................................................................338
7.2 Real-Time Services........................................................................................................................................................338
7.2.1 Deploying a Model as a Real-Time Service........................................................................................................338
7.2.2 Viewing Service Details........................................................................................................................................342
7.2.3 Testing a Service....................................................................................................................................................343
7.2.4 Accessing a Real-Time Service (Token-based Authentication)........................................................................345
7.2.5 Accessing a Real-Time Service (AK/SK-based Authentication).......................................................................350
7.2.6 Accessing a Real-Time Service (Application Authentication)........................................................................357
7.2.7 Integrating a Real-Time Service..........................................................................................................................366
7.2.8 Collecting Data...................................................................................................................................................366
7.3 Batch Services.......................................................................................................................................................375
7.3.1 Deploying a Model as a Batch Service..................................................................................................................375
7.3.2 Viewing the Batch Service Prediction Result..................................................................................................381
7.3.3 Collecting Data...................................................................................................................................................381
7.4 Modifying a Service................................................................................................................................................386
7.5 Starting or Stopping a Service.................................................................................................................................387
7.6 Deleting a Service................................................................................................................................................387

8 Resource Pools..........................................................................................................................................................389

9 Custom Images...........................................................................................................................................................393
9.1 Custom Image Overview.......................................................................................................................................393
9.2 Creating and Uploading a Custom Image...........................................................................................................393
9.3 For Training Models..............................................................................................................................................394
9.3.1 Specifications for Custom Images Used for Training Jobs..............................................................................394
9.3.2 Creating a Training Job Using a Custom Image...............................................................................................398
9.3.3 Example: Creating a Training Job Using a Custom Image..............................................................................401
9.4 For Importing Models........................................................................................................................................404
9.4.1 Specifications for Custom Images Used for Importing Models .................................................. 405
9.4.2 Importing a Model Using a Custom Image .................................................................................... 406
9.4.3 Example: Importing a Model Using a Custom Image ..................................................................... 407

10 Model Package Specifications.............................................................................................................. 412
10.1 Model Package Specifications ........................................................................................................... 412
10.2 Specifications for Compiling the Model Configuration File .............................................................. 414
10.3 Specifications for Compiling Model Inference Code ........................................................................... 428

11 Model Templates.................................................................................................................................................. 434
11.1 Model Template Overview ...................................................................................................................... 434
11.2 Template Description ............................................................................................................................... 436
11.2.1 TensorFlow-Based Image Classification Template ........................................................................ 436
11.2.2 TensorFlow-py27 General Template ................................................................................................. 437
11.2.3 TensorFlow-py36 General Template ................................................................................................. 438
11.2.4 MXNet-py27 General Template ....................................................................................................... 439
11.2.5 MXNet-py36 General Template ....................................................................................................... 440
11.2.6 PyTorch-py27 General Template .................................................................................................... 441
11.2.7 PyTorch-py36 General Template .................................................................................................... 442
11.2.8 Caffe-CPU-py27 General Template ................................................................................................. 443
11.2.9 Caffe-GPU-py27 General Template .................................................................................................. 444
11.2.10 Caffe-CPU-py36 General Template ............................................................................................... 445
11.2.11 Caffe-GPU-py36 General Template ............................................................................................... 446
11.2.12 ARM-Ascend Template .................................................................................................................. 447
11.3 Input and Output Modes ....................................................................................................................... 448
11.3.1 Built-in Object Detection Mode ....................................................................................................... 448
11.3.2 Built-in Image Processing Mode ....................................................................................................... 450
11.3.3 Built-in Predictive Analytics Mode .................................................................................................. 450
11.3.4 Undefined Mode .............................................................................................................................. 453

12 Examples of Custom Scripts......................................................................................................................... 454
12.1 TensorFlow ............................................................................................................................................... 454
12.2 TensorFlow 2.1 ....................................................................................................................................... 460
12.3 PyTorch .................................................................................................................................................. 461
12.4 Caffe ..................................................................................................................................................... 464
12.5 XGBoost ............................................................................................................................................... 470
12.6 PySpark ............................................................................................................................................... 471
12.7 Scikit Learn .......................................................................................................................................... 472

13 Permissions Management ............................................................................................................................. 474
13.1 Creating a User and Granting Permissions ......................................................................................... 474
13.2 Creating a Custom Policy ...................................................................................................................... 476

14 Monitoring .................................................................................................................................................... 478
14.1 ModelArts Metrics ............................................................................................................................... 478
14.2 Setting Alarm Rules.........................................................................................................................................................481
14.3 Viewing Monitoring Metrics.........................................................................................................................................482

15 Audit Logs.............................................................................................................................................................................483
15.1 Key Operations Recorded by CTS............................................................................................................................... 483
15.2 Viewing Audit Logs..........................................................................................................................................................488

A Change History......................................................................................................................................................................490
ModelArts provides online code compiling environments as well as AI Development Lifecycle that covers data preparation, model training, model management, and service deployment for developers who are familiar with code compilation, debugging, and common AI engines, helping the engineers build models efficiently and quickly.

This document describes how to perform AI development on the ModelArts management console. If you use the APIs or SDKs for development, you are advised to view the ModelArts SDK Reference or ModelArts API Reference.

To view the examples of AI development lifecycle, see Modeling with MXNet and Modeling with Notebook. For details about how to use a built-in algorithm to build a model, see AI Beginners: Using a Built-in Algorithm to Build a Model.

**AI Development Lifecycle**

The AI Development Lifecycle function provided by ModelArts takes developers' habits into consideration and provides a variety of engines and scenarios for developers to choose. The following describes the entire process from data preparation to service development using the ModelArts platform.
**Figure 1-1 Process of using ModelArts**

![Diagram showing the process of using ModelArts](image)

**Table 1-1 Process description**

<table>
<thead>
<tr>
<th>Task</th>
<th>Sub Task</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepare Data</td>
<td>Create a dataset.</td>
<td>Create a dataset in ModelArts to manage and preprocess your business data.</td>
<td>Creating a Dataset</td>
</tr>
<tr>
<td></td>
<td>Label data.</td>
<td>Label and preprocess the data in your dataset based on the business logic to facilitate subsequent training. Data labeling affects the model training performance.</td>
<td>Labeling Data</td>
</tr>
<tr>
<td></td>
<td>Publish the dataset.</td>
<td>After labeling data, publish the database to generate a dataset version that can be used for model training.</td>
<td>Publishing a Dataset</td>
</tr>
<tr>
<td>Develop Script</td>
<td>Create a notebook instance.</td>
<td>Create a notebook instance as the development environment.</td>
<td>Creating a Notebook Instance</td>
</tr>
<tr>
<td></td>
<td>Compile code.</td>
<td>Compile code in an existing notebook to directly build a model.</td>
<td>Common Operations on Jupyter Notebook JupyterLab Overview and Common Operations</td>
</tr>
</tbody>
</table>

---

Copyright © Huawei Technologies Co., Ltd.
<table>
<thead>
<tr>
<th>Task</th>
<th>Sub Task</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Export</td>
<td>Export the .py file.</td>
<td>Export the compiled training script as a .py file for subsequent operations, such as model training and management.</td>
<td>Using the Convert to Python File Function</td>
</tr>
<tr>
<td>Train a Model</td>
<td>Create a training job.</td>
<td>Create a training job, and upload and use the compiled training script. After training is complete, a model is generated and stored in OBS.</td>
<td>Creating a Training Job</td>
</tr>
<tr>
<td>(Optional) Create a visualization job.</td>
<td>Create a visualization job (TensorBoard type) to view the model training process, learn about the model, and adjust and optimize the model. Currently, visualization jobs only support the MXNet and TensorFlow engines.</td>
<td>Managing a TensorBoard Job</td>
<td></td>
</tr>
<tr>
<td>Manag e Models</td>
<td>Compile inference code and configuration files.</td>
<td>Following the model package specifications provided by ModelArts, compile inference code and configuration files for your model, and save the inference code and configuration files to the training output location.</td>
<td>Model Package Specifications</td>
</tr>
<tr>
<td>Import the model.</td>
<td>Import the training model to ModelArts to facilitate service deployment.</td>
<td>Importing a Model</td>
<td></td>
</tr>
<tr>
<td>Deploy a Model</td>
<td>Deploy a model as a service.</td>
<td>Deploy a model as a real-time service or a batch service.</td>
<td></td>
</tr>
<tr>
<td>Access the service.</td>
<td>If the model is deployed as a real-time service, you can access and use the service. If the model is deployed as a batch service, you can view the prediction result.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using Built-in Algorithms to Build Models

AI beginners with certain AI knowledge can use their own business data and select common algorithms (ModelArts built-in algorithms) for model training to obtain new models.
2 Data Management

2.1 Data Management Overview

In ModelArts, you can import and label data on the Data Management page to prepare for model building. ModelArts uses datasets as the basis for model development or training.

Dataset Types

ModelArts supports datasets of images, audio, text, tables, videos, and other types for the following purposes:

- **Images**
  - Image classification: identifies a class of objects in images.
  - Object detection: identifies the position and class of each object in an image.

- **Audio**
  - Sound classification: classifies and identifies different sounds.
  - Speech labeling: labels speech content.
  - Speech paragraph labeling: segments and labels speech content.

- **Text**
  - Text classification: assigns labels to text according to its content.
  - Named entity recognition: assigns labels to named entities in text, such as time and locations.
  - Text triplet: assigns labels to entity segments and entity relationships in the text.

- **Tables**
  - Table: applies to structured data processing such as tables. The file format can be CSV or Carbon. You can preview a maximum of 100 records in a table.

- **Videos**
  - Video labeling: identifies the position and class of each object in a video. Currently, only the MP4 format is supported.
• Others
  - Free format: manages data in any format. Currently, labeling is not available for data of the free format type. The free format type is applicable to scenarios where labeling is not required or developers customize labeling. If your dataset needs to contain data in multiple formats or your data format does not meet the requirements of other types of datasets, you can select a dataset in free format.

Figure 2-1 Example of a dataset in free format

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>image003_15941069724.jpg</td>
<td>90.41 KB</td>
<td>jpg</td>
</tr>
<tr>
<td>image005_159410698161.gif</td>
<td>341.29 KB</td>
<td>gif</td>
</tr>
<tr>
<td>lrz_1592801309912.csv</td>
<td>2.36 KB</td>
<td>csv</td>
</tr>
<tr>
<td>train_1592801309925.csv</td>
<td>8.26 KB</td>
<td>csv</td>
</tr>
<tr>
<td>vedio_159410604159.mp4</td>
<td>420.71 KB</td>
<td>mp4</td>
</tr>
</tbody>
</table>

Dataset Management Process and Functions

Figure 2-2 Labeling management process

Table 2-1 Function description

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating a Dataset</td>
<td>Create a dataset.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Image Classification</td>
<td>Label data based on the types of datasets. Data labeling is not supported for datasets in free format or table format.</td>
</tr>
<tr>
<td>Object Detection</td>
<td></td>
</tr>
<tr>
<td>Text Classification</td>
<td></td>
</tr>
<tr>
<td>Named Entity Recognition</td>
<td></td>
</tr>
<tr>
<td>Text Triplet</td>
<td></td>
</tr>
<tr>
<td>Sound Classification</td>
<td></td>
</tr>
<tr>
<td>Speech Labeling</td>
<td></td>
</tr>
<tr>
<td>Speech Paragraph Labeling</td>
<td></td>
</tr>
<tr>
<td>Video Labeling</td>
<td></td>
</tr>
<tr>
<td>Import Operation</td>
<td>Import the local manifest file or data stored in OBS to the dataset.</td>
</tr>
<tr>
<td>Exporting Data</td>
<td>Export part of the data as a new dataset or to OBS. Historical tasks can be viewed and managed.</td>
</tr>
<tr>
<td>Modifying a Dataset</td>
<td>Modify the basic information about a dataset, such as the dataset name, description, and labels.</td>
</tr>
<tr>
<td>Publishing a Dataset</td>
<td>Publish the labeled dataset as a new version for model building.</td>
</tr>
<tr>
<td>Managing Dataset Versions</td>
<td>View data version updates.</td>
</tr>
<tr>
<td>Auto Labeling</td>
<td>quickly label unlabeled data, reducing labeling time by 70%</td>
</tr>
<tr>
<td>Auto Grouping</td>
<td>Enable auto grouping to improve data labeling efficiency.</td>
</tr>
<tr>
<td>Data Features</td>
<td>Analyze data features to help you understand data.</td>
</tr>
<tr>
<td>Deploying a Model in One Click</td>
<td>Use the one-click model deployment function to train a model with the labeled dataset, import the model, and deploy the model as a real-time service all in one step.</td>
</tr>
<tr>
<td>Team Labeling Overview</td>
<td>Allow multiple users to label the same dataset and enable the dataset creator to manage labeling tasks in a unified manner. Add a team and its members to participate in labeling datasets.</td>
</tr>
<tr>
<td>Deleting a Dataset</td>
<td>Delete a dataset to release resources.</td>
</tr>
</tbody>
</table>
Functions Supported by Different Types of Datasets

Different types of datasets provide different functions. For details, see Table 2-2.

<table>
<thead>
<tr>
<th>Dataset Type</th>
<th>Creating a Dataset</th>
<th>Importing Data</th>
<th>Exporting Data</th>
<th>Publishing a Dataset</th>
<th>Modifying a Dataset</th>
<th>Managing Dataset Versions</th>
<th>Auto Labeling</th>
<th>Team Labeling</th>
<th>Auto Grouping</th>
<th>Data Features</th>
<th>Deploying a Model in One Click</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image classification</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Object detection</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Sound classification</td>
<td>Supported</td>
<td>Supported</td>
<td>-</td>
<td>Supported</td>
<td>Supported</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Speech labeling</td>
<td>Supported</td>
<td>Supported</td>
<td>-</td>
<td>Supported</td>
<td>Supported</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Speech paragraph labeling</td>
<td>Supported</td>
<td>Supported</td>
<td>-</td>
<td>Supported</td>
<td>Supported</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Text classification</td>
<td>Supported</td>
<td>Supported</td>
<td>-</td>
<td>Supported</td>
<td>Supported</td>
<td>-</td>
<td>Supported</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2-2 Functions supported by different types of datasets
### 2.2 Creating a Dataset

To manage data using ModelArts, you need to create a dataset first. Then you can perform operations on the dataset, such as labeling data, importing data, and publishing the dataset.

**Prerequisites**

- Before using the data management function, you need permissions to access OBS. This function cannot be used if you are not authorized to access OBS. Before using the data management function, go to the **Settings** page and complete access authorization using an agency.
- You have created OBS buckets and folders for storing data. In addition, the OBS buckets and ModelArts are in the same region.
- You have uploaded data to be used to OBS.
Procedure

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.

2. Click **Create Dataset**. On the **Create Dataset** page, create datasets of different types based on the data type and data labeling requirements.

   a. Enter basic information, the name and description of the dataset.

   ![Figure 2-3 Basic information about a dataset](image)

   b. Select a labeling scene and type as required. For details about the types supported by ModelArts, see **Dataset Types**.

   ![Figure 2-4 Selecting a labeling scene and type](image)

   c. Set the parameters based on the dataset type. For details, see the parameter description of the following dataset types:

      - **Images (Image Classification and Object Detection)**

      - **Audio (Sound Classification, Speech Labeling, and Speech Paragraph Labeling)**

      - **Text (Text Classification, Named Entity Recognition, and Text Triplet)**

      - **Table**

      - **Video**

      - **Other (Free Format)**

   d. Click **Create** in the lower right corner of the page.

   After the dataset is created, the dataset management page is displayed. You can perform the following operations on the dataset: label data, publish dataset versions, manage dataset versions, modify the dataset, import data, and delete the dataset. For details about the operations...
supported by different types of datasets, see Functions Supported by Different Types of Datasets.

Images (Image Classification and Object Detection)

Figure 2-5 Parameters of datasets for image classification and object detection

Table 2-3 Dataset parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Dataset Path</td>
<td>Select the OBS path to the input dataset.</td>
</tr>
<tr>
<td>Output Dataset Path</td>
<td>Select the OBS path to the output dataset.</td>
</tr>
<tr>
<td>NOTE</td>
<td>The output dataset path cannot be the same as the input dataset path or cannot be the subdirectory of the input dataset path. Select an empty directory as the Output Dataset Path.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Label Set     | - **Label Name**: Enter a label name. The label name can contain only Chinese characters, letters, digits, underscores ( _ ), and hyphens ( - ). The name contains 1 to 32 characters.  
  - **Add Label**: Click **Add Label** to add more labels.  
  - Setting a label color: This function is available only for datasets of the object detection type. Select a color from the color palette on the right of a label, or enter the hexadecimal color code to set the color.  
  - Setting label attributes: For an object detection dataset, you can click the plus sign (+) on the right to add label attributes after setting a label color. Label attributes are used to distinguish different attributes of the objects with the same label. For example, yellow kittens and black kittens have the same label *cat* and their label attribute is *color*. |
| Team Labeling | Enable or disable team labeling.  
  To enable the team labeling function, you need to enter the name and type of the team labeling task, and select the labeling team and team members. For details about the parameter settings, see [Creating Team Labeling Tasks](#).  
  Before enabling the team labeling function, ensure that you have added a team and members on the **Labeling Teams** page. If no labeling team is available, click the link on the page to go to the **Labeling Teams** page, and add your team and members. For details, see [Team Labeling Overview](#).  
  After a dataset is created with team labeling enabled, you can view the **Team Labeling** mark in **Labeling Type**. |
Audio (Sound Classification, Speech Labeling, and Speech Paragraph Labeling)

Figure 2-6 Parameters of datasets for sound classification, speech labeling, and speech paragraph labeling

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Dataset Path</td>
<td>Select the OBS path to the input dataset.</td>
</tr>
</tbody>
</table>
| Output Dataset Path               | Select the OBS path to the output dataset. **NOTE**  
The output dataset path cannot be the same as the input dataset path or cannot be the subdirectory of the input dataset path. Select an empty directory as the Output Dataset Path. |
| Label Set (Sound Classification)  | You need to set labels only for datasets of the sound classification type.  
  - **Label Name:** Enter a label name. The label name can contain only Chinese characters, letters, digits, underscores (_), and hyphens (-). The name contains 1 to 32 characters.  
  - **Add Label:** Click Add Label to add more labels. |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Label Management (Speech Paragraph Labeling) | Only datasets for speech paragraph labeling support multiple labels.  
- **Single Label**  
  A single label is used to label a piece of audio that has only one class.  
  - **Label Name**: Enter a label name. The label name can contain contains 1 to 32 characters. Only Chinese characters, letters, digits, underscores (_), and hyphens (-) are allowed.  
  - **Label Color**: Set the label color in the **Label Color** column. You can select a color from the color palette or enter a hexadecimal color code to set the color.  
- **Multiple Labels**  
  Multiple labels are suitable for multi-dimensional labeling. For example, you can label a piece of audio as both noise and speech. For speech, you can label the audio with different speakers. You can click **Add Label Class** to add multiple label classes. A label class can contain multiple labels. The label class and name can contain contains 1 to 32 characters. Only Chinese characters, letters, digits, underscores (_), and hyphens (-) are allowed.  
  - **Label Class**: Set a label class.  
  - **Label Name**: Enter a label name.  
  - **Add Label**: Click **Add Label** to add more labels. |
| Speech Labeling (Speech Paragraph Labeling) | This function is available only for dataset for speech paragraph labeling. By default, it is disabled. If this function is enabled, you can label speech content. |
Text (Text Classification, Named Entity Recognition, and Text Triplet)

**Figure 2-7** Parameters of datasets for text classification, named entity recognition, and text triplet

**Table 2-4** Dataset parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Dataset Path</td>
<td>Select the OBS path to the input dataset.</td>
</tr>
<tr>
<td>Output Dataset Path</td>
<td>Select the OBS path to the output dataset.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong></td>
</tr>
<tr>
<td></td>
<td>The output dataset path cannot be the same as the input dataset path or</td>
</tr>
<tr>
<td></td>
<td>cannot be the subdirectory of the input dataset path. Select an empty</td>
</tr>
<tr>
<td></td>
<td>directory as the <strong>Output Dataset Path</strong>.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| Label Set (for text classification and named entity recognition) | - **Label Name**: Enter a label name. The label name can contain only Chinese characters, letters, digits, underscores (_), and hyphens (-). The name contains 1 to 32 characters.  
- **Add Label**: Click **Add Label** to add more labels.  
- Setting a label color: Select a color from the color palette or enter the hexadecimal color code to set the color. |
| Label Set (for text triplet) | For datasets of the text triplet type, you need to set entity labels and relationship labels.  
- **Entity Label**: You need to set the label name and label color. You can click the plus sign (+) on the right of the color area to add multiple labels.  
- **Relationship Label**: A relationship label is a relationship between two entities. You need to set the source entity and target entity. You need to add at least two entity labels before adding a relationship label. |
| Team Labeling | Enable or disable team labeling.  
To enable the team labeling function, you need to enter the name and type of the team labeling task, and select the labeling team and team members. For details about the parameter settings, see [Creating Team Labeling Tasks](#).  
Before enabling the team labeling function, ensure that you have added a team and members on the **Labeling Teams** page. If no labeling team is available, click the link on the page to go to the **Labeling Teams** page, and add your team and members. For details, see [Team Labeling Overview](#).  
After a dataset is created with team labeling enabled, you can view the **Team Labeling** mark in **Labeling Type**. |
### Table

**Figure 2-8** Parameters of datasets of the table type

![Dataset Parameters Diagram](image-url)

**Table 2-5** Dataset parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Storage Path | Select the OBS path for storing table data. The data imported from the data source is stored in this path. The path cannot be the same as or a subdirectory of the file path in the OBS data source. After a table dataset is created, the following four directories are automatically generated in the storage path:  
- **annotation**: version publishing directory. Each time a version is published, a subdirectory with the same name as the version is generated in this directory.  
- **data**: data storage directory. Imported data is stored in this directory.  
- **logs**: directory for storing logs  
- **temp**: temporary working directory |
<p>| Import      | If you have stored table data on other cloud services, you can enable this function to import data stored on OBS, DLI, or MRS. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Data Source (OBS)** | • **File Path**: Browse all OBS buckets of the account and select the directory where the data file to be imported is located.  
• **Contain Table Header**: If this parameter is enabled, the imported file contains table headers. In this case, the first row of the imported file is used as the column name. Otherwise, the default column name is added and automatically filled in the schema information.  
For details about OBS functions, see the [Object Storage Service Console Operation Guide](#). |
| **Data Source (DWS)** | • **Cluster Name**: The system automatically displays the DWS clusters of the account. You can select a DWS cluster from the drop-down list.  
• **Database Name**: Enter the name of the database where the data is located based on the selected DWS cluster.  
• **Table Name**: Enter the name of the table where the data is located based on the selected database.  
• **Username**: Enter the username of the DWS cluster administrator.  
• **Password**: Enter the password of the DWS cluster administrator.  
For details about DWS functions, see the [Data Warehouse Service User Guide](#).  
**NOTE**  
To import data from DWS, you need to use DLI functions. If you do not have the permission to access DLI, create a DLI agency as prompted. |
| **Data Source (DLI)** | • **Queue Name**: The system automatically displays the DLI queues of the account. You can select a queue from the drop-down list.  
• **Database Name**: All databases are displayed based on the selected queue. Select the required database from the drop-down list.  
• **Table Name**: All tables in the selected database are displayed. Select the required table from the drop-down list.  
For details about DLI functions, see the [Data Lake Insight User Guide](#). |
### Parameter Description

**Data Source (MRS)**

- **Cluster Name**: The system automatically displays the MRS clusters of the account in the list. However, streaming clusters do not support data import. Select the required cluster from the drop-down list.
- **File Path**: Enter the file path based on the selected cluster. The file path is an HDFS path.
- **Contain Table Header**: If this parameter is enabled, the imported file contains table headers.

For details about MRS functions, see the [MapReduce Service User Guide](#).

**Schema**

Names and types of table columns, which must be the same as those of the imported data. Set the column name based on the imported data and select the column type. For details about the supported types, see Table 2-6.

Click **Add Schema** to add a new record. When creating a dataset, you must specify a schema. Once created, the schema cannot be modified.

When data is imported from OBS, the schema of the CSV file in the file path is automatically obtained. If the schemas of multiple CSV files are inconsistent, an error is reported.

### Table 2-6 Migration data types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Storage Space</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>String</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Short</td>
<td>Signed integer</td>
<td>2 bytes</td>
<td>-32768-32767</td>
</tr>
<tr>
<td>Int</td>
<td>Signed integer</td>
<td>4 bytes</td>
<td>-2147483648 to 2147483647</td>
</tr>
<tr>
<td>Long</td>
<td>Signed integer</td>
<td>8 bytes</td>
<td>-9223372036854775808 to 9223372036854775807</td>
</tr>
<tr>
<td>Double</td>
<td>Double-precision floating point</td>
<td>8 bytes</td>
<td>-</td>
</tr>
<tr>
<td>Float</td>
<td>Single-precision floating point</td>
<td>4 bytes</td>
<td>-</td>
</tr>
<tr>
<td>Byte</td>
<td>Signed integer</td>
<td>1 byte</td>
<td>-128-127</td>
</tr>
<tr>
<td>Date</td>
<td>Date type in the format of yyyy-MM-dd, for example, 2014-05-29</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Type</td>
<td>Description</td>
<td>Storage Space</td>
<td>Range</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Timestamp that represents date and time. Format: yyyy-MM-dd HH:mm:ss</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Boolean</td>
<td>Boolean</td>
<td>1 byte</td>
<td>TRUE/FALSE</td>
</tr>
</tbody>
</table>

**Video**

**Figure 2-9** Parameters of datasets of the video type

**Table 2-7** Dataset parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Dataset Path</td>
<td>Select the OBS path to the input dataset.</td>
</tr>
<tr>
<td>Output Dataset Path</td>
<td>Select the OBS path to the output dataset.</td>
</tr>
<tr>
<td>NOTE</td>
<td>The output dataset path cannot be the same as the input dataset path or cannot be the subdirectory of the input dataset path. Select an empty directory as the <strong>Output Dataset Path</strong>.</td>
</tr>
</tbody>
</table>
Parameter | Description
---|---
Label Set |  
- **Label Name**: Enter a label name. The label name can contain only Chinese characters, letters, digits, underscores (_), and hyphens (-). The name contains 1 to 32 characters.
- **Add Label**: Click **Add Label** to add more labels.
- Setting a label color: Select a color from the color palette or enter the hexadecimal color code to set the color.

**Other (Free Format)**

**Figure 2-10** Parameters of datasets of the free format type

![Image of free format parameters](image)

**Table 2-8** Dataset parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Dataset Path</td>
<td>Select the OBS path to the input dataset.</td>
</tr>
<tr>
<td>Output Dataset Path</td>
<td>Select the OBS path to the output dataset.</td>
</tr>
</tbody>
</table>

**NOTE**

The output dataset path cannot be the same as the input dataset path or cannot be the subdirectory of the input dataset path. Select an empty directory as the **Output Dataset Path**.

### 2.3 Labeling Data

#### 2.3.1 Image Classification

Model training uses a large number of labeled images. Therefore, before the model training, add labels to the images that are not labeled. You can add labels...
to images by manual labeling or auto labeling. In addition, you can modify the labels of images, or remove their labels and label the images again.

Before labeling an image in image classification scenarios, you need to understand the following:

- You can add multiple labels to an image.
- A label name can contain a maximum of 32 characters, including Chinese characters, uppercase letters, lowercase letters, digits, hyphens (-), and underscores (_).

**Starting Labeling**

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management** > **Datasets**. The **Datasets** page is displayed.
2. In the dataset list, select the dataset to be labeled based on the labeling type, and click the dataset name to go to the **Dashboard** tab page of the dataset.
   By default, the **Dashboard** tab page of the current dataset version is displayed. If you need to label the dataset of another version, click the **Versions** tab and then click **Set to Current Version** in the right pane. For details, see **Managing Dataset Versions**.
3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed. By default, all data of the dataset is displayed on the dataset details page.

**Synchronizing Data Sources**

ModelArts automatically synchronizes data and labeling information from **Input Dataset Path** to the dataset details page.

For an image classification dataset, the .txt file with the same name in the same directory as the data source is used as the label of the corresponding image. For an object detection dataset, the .xml file with the same name in the same directory is used as the label of the corresponding image.

To quickly obtain the latest data in the OBS bucket, on the All or Unlabeled tab page of the dataset details page, click **Synchronize Data Source** to add data from OBS to the dataset.

**Filtering Data**

On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed, showing all data in the dataset by default. On the **All**, **Unlabeled**, or **Labeled** tab page, you can add filter criteria in the filter criteria area to quickly filter out data you want to view.

The following filter criteria are supported. You can set one or more filter criteria.

- **Example Type**: Select **Hard example** or **Non-hard example**.
- **Label**: Select **All** or one or more labels you specified.
- **Sample Creation Time**: Select **Within 1 month**, **Within 1 day**, or **Custom** to customize a time range.
- **File Name** or **Path**: Filter files by file name or file storage path.
- **Labeled By**: Select the name of the user who performs the labeling operation.
- **Sample Attribute**: Select the attribute generated by auto grouping. This filter criterion can be used only after **auto grouping** is enabled.
- **Data Attribute**: If data collection is enabled for your service and **data is synchronized to the dataset**, you can choose **Data Source > Inference** to filter the data collected from the data collection tasks.

**Figure 2-11 Filter criteria**

<table>
<thead>
<tr>
<th>Filter Criteria</th>
<th>No filter criteria selected.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Type</td>
<td>Hard example</td>
</tr>
<tr>
<td>Label</td>
<td>All dog</td>
</tr>
<tr>
<td>Sample Creation Time</td>
<td>Within 1 month</td>
</tr>
<tr>
<td>File Name</td>
<td>Enter a keyword and press Enter to create a filter criterion.</td>
</tr>
<tr>
<td>Labeled By</td>
<td>Select an annotator.</td>
</tr>
<tr>
<td>Sample Attribute</td>
<td>Select an attribute.</td>
</tr>
<tr>
<td>Data Attribute</td>
<td>Select an attribute.</td>
</tr>
</tbody>
</table>

**Labeling Images (Manually)**

The dataset details page displays images on the **All**, **Labeled**, and **Unlabeled** tabs. Images on the **All** tab page are displayed by default. Click an image to preview it. For the images that have been labeled, the label information is displayed at the bottom of the preview page.

1. On the **Unlabeled** tab page, select the images to be labeled.
   - Manual selection: In the image list, click the selection box in the upper left corner of an image to enter the selection mode, indicating that the image is selected. You can select multiple images of the same type and add labels to them together.
   - Batch selection: If all the images on the current page of the image list belong to the same type, you can click **Select Images on Current Page** in the upper right corner to select all the images on the current page.

2. Add labels to the selected images.
   a. In the label adding area on the right, set the label in the **Label** text box. Click the **Label** text box and select an existing label from the drop-down list. If the existing labels cannot meet the requirements, you can go to the page for **modifying the dataset** and add labels.
   b. Confirm the **Labels of Selected Image** information and click **OK**. The selected image is automatically moved to the **Labeled** tab page. On the **Unlabeled** and **All** tab pages, the labeling information is updated along with the labeling process, including the added label names and the number of images for each label.
**Figure 2-12 Adding labels**

![Adding labels](image)

**Viewing Labeled Images**

On the dataset details page, click the **Labeled** tab to view the list of the labeled images. By default, the corresponding labels are displayed under the image thumbnails. You can also select an image and view the label information of the image in the **File Labels** area on the right.

**Modifying Labeling Information**

After labeling data, you can modify labeled data on the **Labeled** tab page.

- **Modifying based on images**

  On the dataset details page, click the **Labeled** tab, and select one or more images to be modified from the image list. Modify the image information in the label information area on the right.

  Modifying a label: In the **File Labels** area, click the edit icon in the **Operation** column, enter the correct label name in the text box, and click the check mark to complete the modification.

  Deleting a label: In the **File Labels** area, click the delete icon in the **Operation** column to delete the label. This operation deletes only the labels added to the selected image.
Figure 2-13 Modifying a label

<table>
<thead>
<tr>
<th>Label</th>
<th>Count</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>roses</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- **Modifying based on labels**

  On the dataset details page, click the **Labeled** tab. The information about all labels is displayed on the right.
  
  - Modifying a label: Click the editing icon in the **Operation** column. In the displayed dialog box, enter the new label name and click **OK**. After the modification, the images that have been added with the label use the new label name.
  
  - Deleting a label: Click the deletion icon in the **Operation** column. In the displayed dialog box, select **Delete label**, **Delete label and images with only the label (Do not delete source files)**, or **Delete label and images with only the label (Delete source files)**, and click **OK**.

Figure 2-14 Information about all labels

<table>
<thead>
<tr>
<th>Label</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>tulips</td>
<td>0</td>
</tr>
<tr>
<td>roses</td>
<td>0</td>
</tr>
<tr>
<td>sunflowers</td>
<td>0</td>
</tr>
<tr>
<td>dandelion</td>
<td>0</td>
</tr>
<tr>
<td>daisy</td>
<td>0</td>
</tr>
</tbody>
</table>

**Adding Images**

In addition to automatically synchronizing data from **Input Dataset Path**, you can directly add images on ModelArts for data labeling.

1. On the dataset details page, click the **All** or **Unlabeled** tab. Then click **Add**.
2. On the **Add** page that is displayed, click **Add Image**.

Select one or more images to be uploaded in the local environment. Images in JPG, JPEG, PNG, and BMP formats are supported. The size of a single image cannot exceed 5 MB, and the total size of all images uploaded at a time cannot exceed 8 MB.

After the images are selected, their thumbnails and sizes are displayed on the **Add** page.
Figure 2-15 Adding images

Add

Add Image  Selected 1 images (22.32 KB) Clear

Supported image formats are JPG, JPEG, PNG, and BMP. The maximum size of a image is 5 MB. The total size of the files uploaded in one attempt cannot exceed 8 MB.

3. On the Add page, click OK.

The images you have added will be automatically displayed in the image list on the Unlabeled tab page. In addition, the images are automatically saved to the OBS directory specified by Input Dataset Path.

Deleting Images

You can quickly delete the images you want to discard.

On the All, Unlabeled, or Labeled tab page, select the images to be deleted or click Select Images on Current Page to select all images on the page, and click Delete in the upper left corner to delete the images. In the displayed dialog box, select or deselect Delete source files as required. After confirmation, click OK to delete the images.

If a tick is displayed in the upper left corner of an image, the image is selected. If no image is selected on the page, the Delete button is unavailable.

NOTE

If you select Delete source files, images stored in the corresponding OBS directory will be deleted when you delete the selected images. Deleting source files may affect other dataset versions or datasets using those files. As a result, the page display, training, or inference is abnormal. Deleted data cannot be recovered. Exercise caution when performing this operation.
2.3.2 Object Detection

Model training uses a large number of labeled images. Therefore, before the model training, add labels to the images that are not labeled. You can add labels to images by manual labeling or auto labeling. In addition, you can modify the labels of images, or remove their labels and label the images again.

Before labeling an image in object detection scenarios, you need to understand the following:

- All target objects in the image must be labeled.
- Target objects are clear without any blocking and contained within bounding boxes.
- A target object must be entirely contained within a bounding box. The target object cannot exceed the bounding box and no gaps can be left between the box edges and the target object. Otherwise, the background may affect model training.

Starting Labeling

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.
2. In the dataset list, select the dataset to be labeled based on the labeling type, and click the dataset name to go to the Dashboard tab page of the dataset.
   
   By default, the Dashboard tab page of the current dataset version is displayed. If you need to label the dataset of another version, click the Versions tab and then click Set to Current Version in the right pane. For details, see Managing Dataset Versions.
3. On the Dashboard page of the dataset, click Label in the upper right corner. The dataset details page is displayed. By default, all data of the dataset is displayed on the dataset details page.

Synchronizing Data Sources

ModelArts automatically synchronizes data and labeling information from Input Dataset Path to the dataset details page.

For an image classification dataset, the .txt file with the same name in the same directory as the data source is used as the label of the corresponding image. For an object detection dataset, the .xml file with the same name in the same directory is used as the label of the corresponding image.

To quickly obtain the latest data in the OBS bucket, on the All or Unlabeled tab page of the dataset details page, click Synchronize Data Source to add data from OBS to the dataset.

Filtering Data

On the Dashboard tab page of the dataset, the summary of the dataset is displayed by default. In the upper left corner of the page, click Label. The dataset details page is displayed, showing all data in the dataset by default. On the All, Unlabeled, or Labeled tab page, you can add filter criteria in the filter criteria area to quickly filter out data you want to view.
The following filter criteria are supported. You can set one or more filter criteria.

- **Example Type**: Select **Hard example** or **Non-hard example**.
- **Label**: Select **All** or one or more labels you specified.
- **Sample Creation Time**: Select **Within 1 month**, **Within 1 day**, or **Custom** to customize a time range.
- **File Name** or **Path**: Filter files by file name or file storage path.
- **Labeled By**: Select the name of the user who performs the labeling operation.
- **Sample Attribute**: Select the attribute generated by auto grouping. This filter criterion can be used only after **auto grouping** is enabled.
- **Data Attribute**: If data collection is enabled for your service and **data is synchronized to the dataset**, you can choose **Data Source** > **Inference** to filter the data collected from the data collection tasks.

**Figure 2-16 Filter criteria**

![Filter criteria](image)

**Labeling Images (Manually)**

The dataset details page provides the **Labeled** and **Unlabeled** tabs. The **All** tab page is displayed by default.

1. On the **Unlabeled** tab page, click an image. The image labeling page is displayed. For details about how to use common buttons on the **Labeled** tab page, see Table 2-10.

2. In the left tool bar, select a proper labeling shape. The default labeling shape is a rectangle. In this example, the rectangle is used for labeling.

**NOTE**

On the left of the page, multiple tools are provided for you to label images. However, you can use only one tool at a time.
Table 2-9 Supported bounding box

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Rectangle Icon]</td>
<td>Rectangle. Click the edge of the upper left corner of the object to be labeled. A rectangle will be displayed. Drag the rectangle to cover the object and click to label the object.</td>
</tr>
<tr>
<td>![Polygon Icon]</td>
<td>Polygon. In the area where the object to be labeled is located, click to label a point, move the mouse and click multiple points along the edge of the object, and then click the first point again. All the points form a polygon. Therefore, the object to be labeled is in the bounding box.</td>
</tr>
<tr>
<td>![Circle Icon]</td>
<td>Circle. Click the center point of an object, and move the mouse to draw a circle to cover the object and click to label the object.</td>
</tr>
<tr>
<td>![Straight Line Icon]</td>
<td>Straight line. Click to specify the start and end points of an object, and move the mouse to draw a straight line to cover the object and click to label the object.</td>
</tr>
<tr>
<td>![Dotted Line Icon]</td>
<td>Dotted line. Click to specify the start and end points of an object, and move the mouse to draw a dotted line to cover the object and click to label the object.</td>
</tr>
<tr>
<td>![Point Icon]</td>
<td>Point. Click the object in an image to label a point.</td>
</tr>
</tbody>
</table>

3. In the **Add Label** text box, enter a new label name, select the label color, and click **Add**. Alternatively, select an existing label from the drop-down list. Label all objects in an image. Multiple labels can be added to an image. After labeling an image, you can click the image list below the image to quickly select other images that are not labeled and label them on the labeling page.

Figure 2-17 Adding an object detection label
4. Click **Back to Data Labeling Preview** in the upper part of the page to view the labeling information. In the dialog box that is displayed, click **OK** to save the labeling settings.

The selected image is automatically moved to the **Labeled** tab page. On the **Unlabeled** and **All** tab pages, the labeling information is updated along with the labeling process, including the added label names and the number of images for each label.

<table>
<thead>
<tr>
<th>Table 2-10 Common icons on the labeling page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Icon</strong></td>
</tr>
<tr>
<td>←</td>
</tr>
<tr>
<td>→</td>
</tr>
<tr>
<td>‡</td>
</tr>
<tr>
<td>†</td>
</tr>
<tr>
<td>✻</td>
</tr>
<tr>
<td>‡</td>
</tr>
<tr>
<td>‡</td>
</tr>
<tr>
<td>❌</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Viewing Labeled Images**

On the dataset details page, click the **Labeled** tab to view the list of the labeled images. You can click an image to view the label information about the image in the **All Labels** area on the right.

**Modifying Labeling Information**

After labeling data, you can modify labeled data on the **Labeled** tab page.

- **Modifying based on images**

  On the dataset details page, click the **Labeled** tab, select the images to be modified, and click the images. The labeling page is displayed. Modify the image information in the label information area on the right.

  - Modifying a label: In the **Labeling** area, click the edit icon, enter the correct label name in the text box, and click the check mark to complete the modification. Alternatively, click a label. In the image labeling area,
adjust the position and size of the bounding box. After the adjustment is complete, click another label to save the modification.

- Deleting a label: In the Labeling area, click the deletion icon to delete a label from the image.

After deleting the label, click Back to Data Labeling Preview in the upper left corner of the page to exit the labeling page. In the dialog box that is displayed, save the modification. After all labels of an image are deleted, the image is displayed on the Unlabeled tab page.

Figure 2-18 Editing an object detection label

![Figure 2-18](image)

- Modifying based on labels

On the dataset details page, click the Labeled tab. The information about all labels is displayed on the right.

- Modifying a label: Click the editing icon in the Operation column. In the dialog box that is displayed, enter the new label name, select the new label color, and click OK. After the modification, the images that have been added with the label use the new label name.

- Deleting a label: Click the deletion icon in the Operation column to delete a label.

Figure 2-19 All labels for object detection

![Figure 2-19](image)

Adding Images

In addition to automatically synchronizing data from Input Dataset Path, you can directly add images on ModelArts for data labeling.

1. On the dataset details page, click the All or Unlabeled tab. Then click Add.
2. On the Add page that is displayed, click Add Image.

Select one or more images to be uploaded in the local environment. Images in JPG, JPEG, PNG, and BMP formats are supported. The size of a single image cannot exceed 5 MB, and the total size of all images uploaded at a time cannot exceed 8 MB.

After the images are selected, their thumbnails and sizes are displayed on the Add page.
3. On the Add page, click OK.

The images you have added will be automatically displayed in the image list on the Unlabeled tab page. In addition, the images are automatically saved to the OBS directory specified by Input Dataset Path.

Deleting Images

You can quickly delete the images you want to discard.

On the All, Unlabeled, or Labeled tab page, select the images to be deleted or click Select Images on Current Page to select all images on the page, and click Delete in the upper left corner to delete the images. In the displayed dialog box, select or deselect Delete source files as required. After confirmation, click OK to delete the images.

If a tick is displayed in the upper left corner of an image, the image is selected. If no image is selected on the page, the Delete button is unavailable.

NOTE

If you select Delete source files, images stored in the corresponding OBS directory will be deleted when you delete the selected images. Deleting source files may affect other dataset versions or datasets using those files. As a result, the page display, training, or inference is abnormal. Deleted data cannot be recovered. Exercise caution when performing this operation.
2.3.3 Text Classification

Model training requires a large amount of labeled data. Therefore, before the model training, add labels to the files that are not labeled. In addition, you can modify, delete, and re-label the labeled text.

Text classification classifies text content based on labels. Before labeling text content, you need to understand the following:

- Text labeling supports multiple labels. That is, you can add multiple labels to a labeling object.
- A label name can contain a maximum of 32 characters, including Chinese characters, uppercase letters, lowercase letters, digits, hyphens (-), and underscores (_).

Starting Labeling

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.
2. In the dataset list, select the dataset to be labeled based on the labeling type, and click the dataset name to go to the Dashboard tab page of the dataset. By default, the Dashboard tab page of the current dataset version is displayed. If you need to label the dataset of another version, click the Versions tab and then click Set to Current Version in the right pane. For details, see Managing Dataset Versions.
3. On the Dashboard page of the dataset, click Label in the upper right corner. The dataset details page is displayed. By default, all data of the dataset is displayed on the dataset details page.

Labeling Content

The dataset details page displays the labeled and unlabeled text files in the dataset. The Unlabeled tab page is displayed by default.

1. On the Unlabeled tab page, the objects to be labeled are listed in the left pane. In the list, click the text object to be labeled, and select a label in the Label Set area in the right pane. Multiple labels can be added to a labeling object.
   You can repeat this operation to select objects and add labels to the objects.
Figure 2-21 Labeling for text classification

2. After all objects are labeled, click Save Current Page at the bottom of the page to complete labeling text files on the Unlabeled tab page.

Adding Labels

- Adding labels on the Unlabeled tab page: Click the plus sign (+) next to Label Set. On the Add Label page that is displayed, add a label name, select a label color, and click OK.

Figure 2-22 Adding a label (1)

- Adding labels on the Labeled tab page: Click the plus sign (+) next to All Labels. On the Add Label page that is displayed, add a label name, select a label color, and click OK.

Figure 2-23 Adding a label (2)
Viewing the Labeled Text

On the dataset details page, click the Labeled tab to view the list of the labeled text. You can also view all labels supported by the dataset in the All Labels area on the right.

Modifying Labeled Data

After labeling data, you can modify labeled data on the Labeled tab page.

- **Modifying based on texts**
  On the dataset details page, click the Labeled tab, and select the text to be modified from the text list.
  In the text list, click the text. When the text background turns blue, the text is selected. If a text file has multiple labels, you can click above a label to delete the label.

- **Modifying based on labels**
  On the dataset details page, click the Labeled tab. The information about all labels is displayed on the right.
  - Batch modification: In the All Labels area, click the editing icon in the Operation column, modify the label name in the text box, select a label color, and click OK.
  - Batch deletion: In the All Labels area, click the deletion icon in the Operation column to delete the label. In the dialog box that is displayed, select Delete label or Delete label and objects with only the label, and click OK.

Adding Files

In addition to automatically synchronizing data from Input Dataset Path, you can directly add text files on ModelArts for data labeling.

1. On the dataset details page, click the Unlabeled tab. Then click Add File.
2. In the displayed Add File dialog box, set the parameters as required and then select the file to be uploaded.
   Select one or more files to be uploaded in the local environment. Only .txt and .csv files are supported. The total size of files uploaded at a time cannot
exceed 8 MB. **Text and Label Separator** and **Label Separator** must be different.

- **Pattern**: Select **Merge text objects and labels** or **Separate text objects and labels**. An example is provided. Determine the mode of the file to be added by referring to the example.

- **Text and Label Separator**: Select **Tab**, **Space**, **Semicolon**, **Comma**, or **Other**. If you select **Other**, enter a separator in the text box on the right.

- **Label Separator**: Select **Tab**, **Space**, **Semicolon**, **Comma**, or **Other**. If you select **Other**, enter a separator in the text box on the right.

**Figure 2-25 Adding a file**

**Add File**

The file to be uploaded must be in TXT or CSV format, and the file size cannot exceed 8 MB. Use line feed characters to separate rows in the file, and each row of data represents a labeling object.

3. In the **Add File** dialog box, click **Upload**. The files you add will be automatically displayed on the **Unlabeled** or **Labeled** tab page.

### Deleting a File

You can quickly delete the files you want to discard.

- On the **Unlabeled** tab page, select the text to be deleted, and click **Delete** in the upper left corner to delete the text.
- On the **Labeled** tab page, select the text to be deleted and click **Delete**. Alternatively, you can tick **Select Images on Current Page** to select all text on the current page and click **Delete** in the upper left corner.

The background of the selected text is blue.

### 2.3.4 Named Entity Recognition

Named entity recognition assigns labels to named entities in text, such as time and locations. Before labeling, you need to understand the following:

- A label name can contain a maximum of 32 characters, including Chinese characters, uppercase letters, lowercase letters, digits, hyphens (-), and underscores (_).

### Starting Labeling

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management** > **Datasets**. The **Datasets** page is displayed.
2. In the dataset list, select the dataset to be labeled based on the labeling type, and click the dataset name to go to the **Dashboard** tab page of the dataset. By default, the **Dashboard** tab page of the current dataset version is displayed. If you need to label the dataset of another version, click the **Versions** tab and then click **Set to Current Version** in the right pane. For details, see **Managing Dataset Versions**.

3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed. By default, all data of the dataset is displayed on the dataset details page.

### Labeling Content

The dataset details page displays the labeled and unlabeled text files in the dataset. The **Unlabeled** tab page is displayed by default.

1. On the **Unlabeled** tab page, the objects to be labeled are listed in the left pane. In the list, click the text object to be labeled, select a part of text displayed under **Label Set** for labeling, and select a label in the **Label Set** area in the right pane. Multiple labels can be added to a labeling object. You can repeat this operation to select objects and add labels to the objects.

   **Figure 2-26** Labeling for named entity recognition

   ![Labeling for named entity recognition](image)

   2. Click **Save Current Page** in the lower part of the page to complete the labeling.

### Adding Labels

- Adding labels on the **Unlabeled** tab page: Click the plus sign (+) next to **Label Set**. On the **Add Label** page that is displayed, add a label name, select a label color, and click **OK**.

   **Figure 2-27** Adding a named entity label (1)

   ![Adding a named entity label](image)

- Adding labels on the **Labeled** tab page: Click the plus sign (+) next to **All Labels**. On the **Add Label** page that is displayed, add a label name, select a label color, and click **OK**.
Viewing the Labeled Text

On the dataset details page, click the **Labeled** tab to view the list of the labeled text. You can also view all labels supported by the dataset in the **All Labels** area on the right.

Modifying Labeled Data

After labeling data, you can modify labeled data on the **Labeled** tab page.

On the dataset details page, click the **Labeled** tab, and modify the text information in the label information area on the right.

- **Modifying based on texts**
  
  On the dataset details page, click the **Labeled** tab, and select the text to be modified from the text list.

  Manual deletion: In the text list, click the text. When the text background turns blue, the text is selected. On the right of the page, click above a text label to delete the label.

- **Modifying based on labels**

  On the dataset details page, click the **Labeled** tab. The information about all labels is displayed on the right.
- Batch modification: In the All Labels area, click the editing icon in the Operation column, add a label name in the text box, select a label color, and click OK.
- Batch deletion: In the All Labels area, click the deletion icon in the Operation column to delete the label. In the dialog box that is displayed, select Delete label or Delete label and objects with only the label, and click OK.

Adding Files

In addition to automatically synchronizing data from Input Dataset Path, you can directly add text files on ModelArts for data labeling.

1. On the dataset details page, click the Unlabeled tab. Then click Add File.
2. In the displayed Add File dialog box, set the parameters as required and then select the file to be uploaded.
   Select one or more files to be uploaded in the local environment. Only .txt and .csv files are supported. The total size of files uploaded at a time cannot exceed 8 MB.

   Figure 2-30 Adding a file

   Add File

   ! The file to be uploaded must be in TXT or CSV format, and the file size cannot exceed 8 MB. Use line feed characters to separate rows in the file, and each row of data represents a labeling object.

   3. In the Add File dialog box, click Upload. The files you add will be automatically displayed on the Unlabeled tab page.

Deleting a File

You can quickly delete the files you want to discard.

- On the Unlabeled tab page, select the text to be deleted, and click Delete in the upper left corner to delete the text.
- On the Labeled tab page, select the text to be deleted and click Delete. Alternatively, you can tick Select Images on Current Page to select all text on the current page and click Delete in the upper left corner.

   The background of the selected text is blue.

2.3.5 Text Triplet

Triplet labeling is suitable for scenarios where structured information, such as subjects, predicates, and objects, needs to be labeled in statements. With this
function, not only entities in statements, but also relationships between entities can be labeled. Triplet labeling is often used in natural language processing tasks such as dependency syntax analysis and information extraction.

Text triplet labeling involves two classes of important labels: Entity Label and Relationship Label. For the Relationship Label, you need to set its Source entity and Target entity.

- You can define multiple entity and relationship labels for a text object.
- The Entity Label defined during dataset creation cannot be deleted.

Precautions

Before labeling, ensure that the Entity Label and Relationship Label of a dataset have been defined. For the Relationship Label, you need to set its Source entity and Target entity. The Relationship Label must be between the defined Source entity and Target entity.

For example, if two entities are labeled as Place, you cannot add any relationship label between them, as shown in Figure 2-31. If a relationship label cannot be added, a red cross is displayed, as shown in Figure 2-32.

![Figure 2-31 Example of entity and relationship labels](image)

![Figure 2-32 Failure of adding a relationship label](image)

Starting Labeling

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.
2. In the dataset list, select the dataset to be labeled based on the labeling type, and click the dataset name to go to the Dashboard tab page of the dataset. By default, the Dashboard tab page of the current dataset version is displayed. If you need to label the dataset of another version, click the Versions tab and then click Set to Current Version in the right pane. For details, see Managing Dataset Versions.
3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed. By default, all data of the dataset is displayed on the dataset details page.

**Labeling Content**

The dataset details page displays the labeled and unlabeled text objects in the dataset. The Unlabeled tab page is displayed by default.

1. On the Unlabeled tab page, the objects to be labeled are listed in the left pane. In the list, click a text object, select the corresponding text content on the right pane, and select an entity name from the displayed entity list to label the content.

   ![Figure 2-33 Labeling an entity](image)

   **Eric** was born in **BeiJing**, but he lives in **HangZhou**.

   - **Person**
   - **Place**

2. After labeling multiple entities, click the source entity and target entity in sequence and select a relationship type from the displayed relationship list to label the relationship.

   ![Figure 2-34 Labeling a relationship](image)

   **Eric** was born in **BeiJing**, but he lives in **HangZhou**.

3. After all objects are labeled, click **Save Current Page** at the bottom of the page.

   **NOTE**

   You cannot modify the labels of a dataset in the text triplet type on the labeling page. Instead, click **Edit** to enter the **Modify Dataset** page and modify the **Entity Label** and **Relationship Label**.

**Modifying Labeled Data**

After labeling data, you can modify labeled data on the Labeled tab page.

On the dataset details page, click the Labeled tab. Select a text object in the left pane and the right pane displays the detailed label information. You can move your cursor to the entity or relationship label, and right-click to delete it. You can also click the source entity and target entity in sequence to add a relationship label.
You can click **Delete Labels on Current Item** at the bottom of the page to delete all labels in the selected text object.

**Figure 2-36 Deleting current labels**

![Figure 2-36 Deleting current labels](image)

### Adding a File

In addition to automatically synchronizing data from **Input Dataset Path**, you can directly add text files on ModelArts for data labeling.

1. On the dataset details page, click the **Unlabeled** tab. Then click **Add File**.
2. In the **Add File** dialog box that is displayed, select the files to be uploaded. Select one or more files to be uploaded in the local environment. Only `.txt` and `.csv` files are supported. The total size of files uploaded at a time cannot exceed 8 MB.

   **Figure 2-37 Add a file to be uploaded**

   ![Add File](image)

   The file to be uploaded must be in TXT or CSV format, and the file size cannot exceed 8 MB. Use line feed characters to separate rows in the file, and each row of data represents a labeling object.

3. In the **Add File** dialog box, click **Upload**. The files you add will be automatically displayed in the **Labeling Objects** list on the **Unlabeled** tab page.

### Deleting a File

You can quickly delete the files you want to discard.

- On the **Unlabeled** tab page, select the text to be deleted, and click **Delete** in the upper left corner to delete the text.
• On the Labeled tab page, select the text to be deleted and click Delete. Alternatively, you can tick Select Images on Current Page to select all text objects on the current page and click Delete in the upper left corner.

The background of the selected text is blue. If no text is selected on the page, the Delete button is unavailable.

2.3.6 Sound Classification

Model training requires a large amount of labeled data. Therefore, before the model training, label the unlabeled audio files. ModelArts enables you to label audio files in batches by one click. In addition, you can modify the labels of audio files, or remove their labels and label the audio files again.

Starting Labeling

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.

2. In the dataset list, select the dataset to be labeled based on the labeling type, and click the dataset name to go to the Dashboard tab page of the dataset. By default, the Dashboard tab page of the current dataset version is displayed. If you need to label the dataset of another version, click the Versions tab and then click Set to Current Version in the right pane. For details, see Managing Dataset Versions.

3. On the Dashboard page of the dataset, click Label in the upper right corner. The dataset details page is displayed. By default, all data of the dataset is displayed on the dataset details page.

Synchronizing the Data Source

ModelArts automatically synchronizes data and labeling information from Input Dataset Path to the dataset details page.

To quickly obtain the latest data in the OBS bucket, click Synchronize Data Source on the Unlabeled tab page of the dataset details page to add the data uploaded using OBS to the dataset.

Labeling Audio Files

The dataset details page displays the labeled and unlabeled audio files. The Unlabeled tab page is displayed by default. Click on the left of the audio to preview the audio.

1. On the Unlabeled tab page, select the audio files to be labeled.
   - Manual selection: In the audio list, click the target audio. If the blue check box is displayed in the upper right corner, the audio is selected. You can select multiple audio files of the same type and label them together.
   - Batch selection: If all audio files of the current page belong to one type, you can click Select Images on Current Page in the upper right corner of the list to select all the audio files on the page.

2. Add labels.
a. In the right pane, set a label name in the Label text box.

   Method 1 (the required label already exists): In the right pane, select a shortcut from the Shortcut drop-down list, select an existing label name from the Label text box, and click OK.

   Method 2 (adding a label): In the right pane, select a shortcut from the Shortcut drop-down list, and enter a new label name in the Label text box.

b. The selected audio files are automatically moved to the Labeled tab page. On the Unlabeled tab page, the labeling information is updated along with the labeling process, including the added label names and the number of audio files corresponding to each label.

   **NOTE**

   **Shortcut key description**: After specifying a shortcut key for a label, you can select an audio file and press the shortcut key to add a label for the audio file. Example: Specify 1 as the shortcut key for the aa label. Select one or more files and press 1 during data labeling. A message is displayed, asking you whether to label the files with aa. Click OK.

   Each label has a shortcut key. A shortcut key cannot be specified for different labels.

   Shortcut keys can greatly improve the labeling efficiency.

**Figure 2-38 Adding an audio label**

- Label: Selected 1 sound files.
  - 1: airplane

**Viewing the Labeled Audio Files**

On the dataset details page, click the Labeled tab to view the list of the labeled audio files. Click an audio file. You can view the label information about the audio file in the File Labels area on the right.

**Modifying Labels**

After labeling data, you can modify labeled data on the Labeled tab page.

- **Modifying based on audio**

  On the data labeling page, click the Labeled tab. Select one or more audio files to be modified from the audio list. Modify the label in the label details area on the right.

  - Modifying a label: In the File Labels area, click the edit icon in the Operation column, enter the correct label name in the text box, and click the check mark to complete the modification.

  - Deleting a label: In the File Labels area, click the delete icon in the Operation column to delete the label.

- **Modifying based on labels**
On the dataset details page, click the **Labeled** tab. The information about all labels is displayed on the right.

**Figure 2-39** Information about all labels

<table>
<thead>
<tr>
<th>Label</th>
<th>Count</th>
<th>Operation</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>airplane</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Modifying a label: Click the editing icon in the **Operation** column. In the dialog box that is displayed, enter the new label name and click **OK**. After the modification, the new label applies to the audio files that contain the original label.
- Deleting a label: Click the deletion icon in the **Operation** column. In the displayed dialog box, select the object to be deleted as prompted and click **OK**.

### Adding Audio Files

In addition to automatically synchronizing data from **Input Dataset Path**, you can directly add audio files on ModelArts for data labeling.

1. On the dataset details page, click the **Unlabeled** tab. Then click **Add Audio** in the upper left corner.
2. In the **Add Audio** dialog box that is displayed, click **Add Audio**. Select the audio files to be uploaded in the local environment. Only WAV audio files are supported. The size of an audio file cannot exceed 4 MB. The total size of audio files uploaded at a time cannot exceed 8 MB.
3. In the **Add Audio** dialog box, click **OK**. The audio files you add will be automatically displayed on the **Unlabeled** tab page. In addition, the audio files are automatically saved to the OBS directory specified by **Input Dataset Path**.

### Deleting Audio Files

You can quickly delete the audio files you want to discard.

On the **Unlabeled** or **Labeled** tab page, select the audio files to be deleted one by one or tick **Select Images on Current Page** to select all audio files on the page, and then click **Delete File** in the upper left corner. In the displayed dialog box, select or deselect **Delete source files** as required. After confirmation, click **OK** to delete the audio files.

If a tick is displayed in the upper right corner of an audio file, the audio file is selected. If no audio file is selected on the page, the **Delete File** button is unavailable.
If you select **Delete source files**, audio files stored in the corresponding OBS directory will be deleted when you delete the selected audio files. Deleting source files may affect other dataset versions or datasets using those files. As a result, the page display, training, or inference is abnormal. Deleted data cannot be recovered. Exercise caution when performing this operation.

### 2.3.7 Speech Labeling

Model training requires a large amount of labeled data. Therefore, before the model training, label the unlabeled audio files. ModelArts enables you to label audio files in batches by one click. In addition, you can modify the labels of audio files, or remove their labels and label the audio files again.

#### Starting Labeling

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.
2. In the dataset list, select the dataset to be labeled based on the labeling type, and click the dataset name to go to the **Dashboard** tab page of the dataset. By default, the **Dashboard** tab page of the current dataset version is displayed. If you need to label the dataset of another version, click the **Versions** tab and then click **Set to Current Version** in the right pane. For details, see **Managing Dataset Versions**.
3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed. By default, all data of the dataset is displayed on the dataset details page.

#### Synchronizing the Data Source

ModelArts automatically synchronizes data and labeling information from **Input Dataset Path** to the dataset details page.

To quickly obtain the latest data in the OBS bucket, click **Synchronize Data Source** on the **Unlabeled** tab page of the dataset details page to add the data uploaded using OBS to the dataset.

#### Labeling Audio Files

The dataset details page displays the labeled and unlabeled audio files. The **Unlabeled** tab page is displayed by default.

1. In the audio file list on the **Unlabeled** tab page, click the target audio file. In the area on the right, the audio file is displayed. Click below the audio file to play the audio.
2. In **Speech Content**, enter the speech content.
3. After entering the content, click **OK** to complete the labeling. The audio file is automatically moved to the **Labeled** tab page.
Viewing the Labeled Audio Files

On the dataset details page, click the Labeled tab to view the list of the labeled audio files. Click the audio file to view the audio content in the Speech Content text box on the right.

Modifying Labeled Data

After labeling data, you can modify labeled data on the Labeled tab page.

On the data labeling page, click the Labeled tab, and select the audio file to be modified from the audio file list. In the label information area on the right, modify the content of the Speech Content text box, and click OK to complete the modification.

Adding Audio Files

In addition to automatically synchronizing data from Input Dataset Path, you can directly add audio files on ModelArts for data labeling.

1. On the dataset details page, click the Unlabeled tab. Then click Add Audio in the upper left corner.
2. In the Add Audio dialog box that is displayed, click Add Audio.
   Select the audio files to be uploaded in the local environment. Only WAV audio files are supported. The size of an audio file cannot exceed 4 MB. The total size of audio files uploaded at a time cannot exceed 8 MB.
3. In the Add Audio dialog box, click OK.
   The audio files you add will be automatically displayed on the Unlabeled tab page. In addition, the audio files are automatically saved to the OBS directory specified by Input Dataset Path.

Deleting Audio Files

You can quickly delete the audio files you want to discard.

On the Unlabeled or Labeled tab page, select the audio files to be deleted, and then click Delete File in the upper left corner. In the displayed dialog box, select or deselect Delete source files as required. After confirmation, click OK to delete the audio files.
If you select **Delete source files**, audio files stored in the corresponding OBS directory will be deleted when you delete the selected audio files. Deleting source files may affect other dataset versions or datasets using those files. As a result, the page display, training, or inference is abnormal. Deleted data cannot be recovered. Exercise caution when performing this operation.

### 2.3.8 Speech Paragraph Labeling

Model training requires a large amount of labeled data. Therefore, before the model training, label the unlabeled audio files. ModelArts enables you to label audio files. In addition, you can modify the labels of audio files, or remove their labels and label the audio files again.

#### Starting Labeling

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.
2. In the dataset list, select the dataset to be labeled based on the labeling type, and click the dataset name to go to the **Dashboard** tab page of the dataset. By default, the **Dashboard** tab page of the current dataset version is displayed. If you need to label the dataset of another version, click the **Versions** tab and then click **Set to Current Version** in the right pane. For details, see [Managing Dataset Versions](#).
3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed. By default, all data of the dataset is displayed on the dataset details page.

#### Synchronizing the Data Source

ModelArts automatically synchronizes data and labeling information from **Input Dataset Path** to the dataset details page.

To quickly obtain the latest data in the OBS bucket, click **Synchronize Data Source** on the **Unlabeled** tab page of the dataset details page to add the data uploaded using OBS to the dataset.

#### Labeling Audio Files

The dataset details page displays the labeled and unlabeled audio files. The **Unlabeled** tab page is displayed by default.

1. In the audio file list on the **Unlabeled** tab page, click the target audio file. In the area on the right, the audio file is displayed. Click below the audio file to play the audio.
2. Select an audio segment based on the content being played, and enter the audio file label and content in the **Speech Content** text box.
Figure 2-41 Labeling an audio file

3. After entering the content, click OK to complete the labeling. The audio file is automatically moved to the Labeled tab page.

Viewing the Labeled Audio Files

On the dataset details page, click the Labeled tab to view the list of the labeled audio files. Click the audio file to view the audio content in the Speech Content text box on the right.

Modifying Labeled Data

After labeling data, you can modify labeled data on the Labeled tab page.

- Modifying a label: On the dataset details page, click the Labeled tab, and select the audio file to be modified from the audio file list. In the Speech Content area, modify Label and Content, and click OK to complete the modification.

- Deleting a label: Click in the Operation column of the target number to delete the label of the audio segment. Alternatively, you can click the cross (x) icon above the labeled audio file to delete the label. Then click OK.

Adding Audio Files

In addition to automatically synchronizing data from Input Dataset Path, you can directly add audio files on ModelArts for data labeling.

1. On the dataset details page, click the Unlabeled tab. Then click Add Audio in the upper left corner.

2. In the Add Audio dialog box that is displayed, click Add Audio.
   Select the audio files to be uploaded in the local environment. Only WAV audio files are supported. The size of an audio file cannot exceed 4 MB. The total size of audio files uploaded at a time cannot exceed 8 MB.

3. In the Add Audio dialog box, click OK.
   The audio files you add will be automatically displayed on the Unlabeled tab page. In addition, the audio files are automatically saved to the OBS directory specified by Input Dataset Path.

Deleting Audio Files

You can quickly delete the audio files you want to discard.

On the Unlabeled or Labeled tab page, select the audio files to be deleted, and then click Delete File in the upper left corner. In the displayed dialog box, select
or deselect **Delete source files** as required. After confirmation, click **OK** to delete the audio files.

**NOTE**

If you select **Delete source files**, audio files stored in the corresponding OBS directory will be deleted when you delete the selected audio files. Deleting source files may affect other dataset versions or datasets using those files. As a result, the page display, training, or inference is abnormal. Deleted data cannot be recovered. Exercise caution when performing this operation.

## 2.3.9 Video Labeling

Model training requires a large amount of labeled video data. Therefore, before the model training, label the unlabeled video files. ModelArts enables you to label video files. In addition, you can modify the labels of video files, or remove their labels and label the video files again.

### Starting Labeling

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.
2. In the dataset list, select the dataset to be labeled based on the labeling type, and click the dataset name to go to the **Dashboard** tab page of the dataset. By default, the **Dashboard** tab page of the current dataset version is displayed. If you need to label the dataset of another version, click the **Versions** tab and then click **Set to Current Version** in the right pane. For details, see **Managing Dataset Versions**.
3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed. By default, all data of the dataset is displayed on the dataset details page.

### Synchronizing Data Sources

ModelArts automatically synchronizes data and labeling information from **Input Dataset Path** to the dataset details page.

To quickly obtain the latest data in the OBS bucket, click **Synchronize Data Source** on the **Unlabeled** tab page of the dataset details page to add the data uploaded using OBS to the dataset.

### Labeling Video Files

On the dataset details page, both unlabeled and labeled video files in the dataset are displayed.

1. On the **Unlabeled** tab page, click the target video file in the video list on the left. The labeling page is displayed.
2. Play the video. When the video is played to the time point to be labeled, click the pause button in the progress bar to pause the video to a specific image.
3. In the left pane, select a bounding box. By default, a rectangular box is selected. Drag the mouse to select an object in the video image, enter a new label name in the displayed **Add Label** text box, select a label color, and click...
**Add** to label the object. Alternatively, select an existing label from the drop-down list and click **Add** to label the object. Label all objects in the image. Multiple labels can be added to an image.

The supported bounding boxes are the same as those supported by Object Detection. For details, see Table 2-9 in Object Detection.

**Figure 2-42** Labeling video files

4. After the previous image is labeled, click the play button on the progress bar to resume the playback. Then, repeat 3 to complete labeling on the entire video.

The labeled time points of the current video are displayed on the right of the page.

**Figure 2-43** File labels

5. Click **Back to Data Labeling Preview** in the upper left corner of the page. The dataset details page is displayed, and the labeled video file is displayed on the **Labeled** tab page.

**Modifying Labeled Data**

After labeling data, you can delete labeled data on the **Labeled** tab page.

- Click ✗ in the **Operation** column of the target number to delete the label of the video segment. Alternatively, you can click the cross (x) icon above the labeled video file to delete the label. Then click **OK**.

On the **Labeled** tab page, click the target video file. In the **All Labels** area on the right of the labeling page, click the triangle icon on the right of the time point to view details. You can modify or delete a label.
Modifying a label: Click the edit icon on the right of a label to modify the label name.

Deleting a label: Click the delete icon on the right of a label to delete the label. If you click the delete icon on the right of the image time, all labels on the image are deleted.

Figure 2-44 Modifying labeled data

Deleting a Video File

You can quickly delete the video files you want to discard.

On the All, Unlabeled, or Labeled tab page, select the video files to be deleted or click Select Images on Current Page to select all video files on the page, and click Delete in the upper left corner to delete the video files. In the displayed dialog box, select or deselect Delete source files as required. After confirmation, click OK to delete the videos.

If a tick is displayed in the upper left corner of a video file, the video file is selected. If no video file is selected on the page, the Delete File button is unavailable.

NOTE

If you select Delete source files, video files stored in the corresponding OBS directory will be deleted when you delete the selected video files. Deleting source files may affect other dataset versions or datasets using those files. As a result, the page display, training, or inference is abnormal. Deleted data cannot be recovered. Exercise caution when performing this operation.

2.4 Importing Data

2.4.1 Import Operation

After a dataset is created, you can directly synchronize data from the dataset. Alternatively, you can import more data by importing the dataset. Currently, data can be imported from an OBS directory or the manifest file.

Prerequisites

- You have created a dataset.
- You have stored the data to be imported in OBS. You have stored the manifest file in OBS.
- The OBS buckets and ModelArts are in the same region.
Import Modes

There are two import modes: **OBS path** and **Manifest file**.

- **OBS path**: indicates that the dataset to be imported has been stored in an OBS directory in advance. In this case, you need to select an OBS path that you can access. In addition, the directory structure in the OBS path must comply with the specifications. For details, see Specifications for Importing Data from an OBS Directory. Only the following types of dataset support the OBS path import mode: Image classification, Object detection, Text classification, Table, and Sound classification.

- **Manifest file**: indicates that the dataset file is in the manifest format and data is imported from the manifest file. The manifest file defines the mapping between labeling objects and content. In addition, the manifest file has been uploaded to OBS. The maximum size of a manifest file is 8 MB. For details about the specifications of the manifest file, see Specifications for Importing the Manifest File.

**NOTE**

Before importing an object detection dataset, ensure that the labeling range of the labeling file does not exceed the size of the original image. Otherwise, the import may fail.

Importing Data from an OBS Path

The parameters on the GUI for data import vary according to the dataset type. The following uses a dataset of the image classification type as an example.

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.
2. Locate the row that contains the desired dataset and choose More > Import in the Operation column.
   Alternatively, you can click the dataset name to go to the Dashboard tab page of the dataset, and click Import in the upper right corner.
3. In the Import dialog box, set Import Mode to **OBS path** and set OBS path to the path for storing data. Then click OK.

**NOTE**

You can import a dataset of the table type from data sources such as OBS, Data Warehouse Service (DWS), Data Lake Insight (DLI), and MapReduce Service (MRS). The settings and data requirements for importing a dataset are the same as those for creating a dataset. For details about the parameters, see **Table**.
After the data import is successful, the data is automatically synchronized to the dataset. On the Datasets page, you can click the dataset name to view its details and label the data.

**Importing Data from a Manifest File**

The parameters on the GUI for data import vary according to the dataset type. The following uses a dataset of the object detection type as an example. Datasets of the table type cannot be imported from the manifest file.

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.

2. Locate the row that contains the desired dataset and choose **More > Import** in the **Operation** column. Alternatively, you can click the dataset name to go to the **Dashboard** tab page of the dataset, and click **Import** in the upper right corner.

3. In the **Import** dialog box, set the parameters as follows and click **OK**.
   - **Import Mode**: Select **Manifest file**.
   - **Manifest file**: Select the OBS path for storing the manifest file.
   - **Import by Label**: The system automatically obtains the labels of the dataset. You can click **Add Label** to add a label or click the deletion icon on the right to delete a label. This field is optional. After importing a dataset, you can add or delete labels during data labeling.
   - **Import labels**: If this parameter is selected, the labels defined in the manifest file are imported to the ModelArts dataset.
   - **Import only hard examples**: If this parameter is selected, only the hard attribute data of the manifest file is imported. Examples whose hard attribute is true in the manifest file are hard examples.
After the data import is successful, the data is automatically synchronized to the dataset. On the Datasets page, you can click the dataset name to go to the Dashboard tab page of the dataset, and click Label in the upper right corner. On the displayed dataset details page, view detailed data and label data.

2.4.2 Specifications for Importing Data from an OBS Directory

When a dataset is imported, the data storage directory and file name must comply with the ModelArts specifications if the data to be used is stored in OBS.

Only the following types of dataset support the OBS path import mode: Image classification, Object detection, Text classification, Table, and Sound classification. A dataset of the table type can be imported from data sources such as OBS, DWS, DLI, and MRS.

**NOTE**

To import data from an OBS directory, you must have the read permission on the OBS directory.

**Image Classification**

- Image classification data can be in two modes. The first mode (directory mode) supports only single labels. The second mode (.txt label files) supports multiple labels.
  - Images with the same label must be stored in the same directory, and the label name is the directory name. If there are multiple levels of directories, the last level is used as the label name.

In the following example, Cat and Dog are label names.

```
dataset-import-example
├── Cat
│   ├── 10.jpg
│   └── 11.jpg
│           12.jpg
└── Dog
    └── 1.jpg
```
– If .txt files exist in the directory, the content in the .txt files is used as the image label. This mode is better than the previous one.

In the following example, import-dir-1 and import-dir-2 are the imported subdirectories:

```
dataset-import-example
  └── import-dir-1
      ├── 10.jpg
      │    10.txt
      │    11.jpg
      │    11.txt
      │    12.jpg
      │    12.txt
      └── import-dir-2
          ├── 1.jpg
          │    1.txt
          │    2.jpg
          │    2.txt
```

The following shows a label file for a single label, for example, the 1.txt file:

```
Cat
```

The following shows a label file for multiple labels, for example, the 1.txt file:

```
Cat
Dog
```

- Only images in JPG, JPEG, PNG, and BMP formats are supported. The size of a single image cannot exceed 5 MB, and the total size of all images uploaded at a time cannot exceed 8 MB.

### Object Detection

- The simple mode of object detection requires users store labeled objects and their label files (in one-to-one relationship with the labeled objects) in the same directory. For example, if the name of the labeled object file is `IMG_20180919_114745.jpg`, the name of the label file must be `IMG_20180919_114745.xml`.

The label files for object detection must be in PASCAL VOC format. For details about the format, see Table 2-16.

Example:

```
├── dataset-import-example
│    └── IMG_20180919_114732.jpg
│    └── IMG_20180919_114732.xml
│    └── IMG_20180919_114745.jpg
│    └── IMG_20180919_114745.xml
│    └── IMG_20180919_114945.jpg
│    └── IMG_20180919_114945.xml
```

A label file example is as follows:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<annotation>
  <folder>NA</folder>
  <filename>bike_1_1593531469339.png</filename>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>554</width>
    <height>606</height>
```

● Only images in JPG, JPEG, PNG, and BMP formats are supported. The size of a single image cannot exceed 5 MB, and the total size of all images uploaded at a time cannot exceed 8 MB.

**Text Classification**

Text classification supports two import modes.

- The labeled objects and labels for text classification are in the same text file. You can specify a separator to separate the labeled objects and labels, as well as multiple labeled objects.

  For example, the following shows an example text file. The **Tab** key is used to separate the labeled object from the label.

  It touches good and responds quickly. I don’t know how it performs in the future.  
  **positive**

  Three months ago, I bought a very good phone and replaced my old one with it. It can operate longer between charges.  
  **positive**

  Why does my phone heat up if I charge it for a while? The volume button stuck after being pressed down.  
  **negative**

  It's a gift for Father’s Day. The logistics is fast and I received it in 24 hours. I like the earphones because the bass sounds feel good and they would not fall off.  
  **positive**

- The labeled objects and label files for text classification are text files, and correspond to each other based on the rows. For example, the first row in a label file indicates the label of the first row in the file of the labeled object.

  For example, the content of labeled object **COMMENTS_20180919_114745.txt** is as follows:

  It touches good and responds quickly. I don’t know how it performs in the future.

  Three months ago, I bought a very good phone and replaced my old one with it. It can operate longer between charges.

  Why does my phone heat up if I charge it for a while? The volume button stuck after being pressed down.

  It's a gift for Father’s Day. The logistics is fast and I received it in 24 hours. I like the earphones because the bass sounds feel good and they would not fall off.
The content of label file `COMMENTS_20180919_114745_result.txt` is as follows:

positive
negative
negative
positive

The data format requires users to store labeled objects and their label files (in one-to-one relationship with the labeled objects) in the same directory. For example, if the name of the labeled object file is `COMMENTS_20180919_114745.txt`, the name of the label file must be `COMMENTS_20180919_114745_result.txt`.

Example of data file storage:

```
│ dataset-import-example
│    COMMENTS_20180919_114732.txt
│    COMMENTS_20180919_114732_result.txt
│    COMMENTS_20180919_114745.txt
│    COMMENTS_20180919_114745_result.txt
│    COMMENTS_20180919_114945.txt
│    COMMENTS_20180919_114945_result.txt
```

**Sound Classification**

For sound classification, sound files with the same label must be stored in the same directory, and the label name is the directory name.

Example:

```
│ dataset-import-example
│    Cat
│        10.wav
│        11.wav
│        12.wav
│    Dog
│        1.wav
│        2.wav
│        3.wav
```

**Table**

You can import data from OBS, DWS, DLI, and MRS.

Import description:

1. The prerequisite for successful import is that the schema of the data source must be the same as that specified during dataset creation. The schema indicates column names and types of a table. Once specified during dataset creation, the values cannot be changed.

2. If the data format is invalid, the data is set to null values. For details, see Table 2-6.

3. When a CSV file is imported from OBS or MRS, the data type is not verified, but the number of columns must be the same as that in the schema of the dataset.

The following describes how to import data from different data sources:

- Importing data from OBS

CSV files can be imported from OBS. You need to select the directory where the files are stored. The number of columns in the CSV file must be the same
as that in the dataset schema. The schema of the CSV file can be automatically obtained.

<table>
<thead>
<tr>
<th>dataset-import-example</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_import_1.csv</td>
</tr>
<tr>
<td>table_import_2.csv</td>
</tr>
<tr>
<td>table_import_3.csv</td>
</tr>
<tr>
<td>table_import_4.csv</td>
</tr>
</tbody>
</table>

- **Importing data from DWS**
  To import data from DWS, you need to select a DWS cluster and enter the database name, table name, username, and password. The schema (column name and type) of the imported table must be the same as that of the dataset.

- **Importing data from DLI**
  To import data from DLI, you need to select the DLI queue, database, and table name. The schema (column name and type) of the selected table must be the same as that of the dataset. The schema of the selected table can be automatically obtained. The default queue of DLI is used only for experience. Different accounts may preempt resources. Therefore, resources need to be queued. You may not be able to obtain required resources each time to perform related operations. DLI supports schema mapping. That is, the schema field name of the imported table can be different from that of the dataset, but the type must be the same.

- **Importing data from MRS**
  Data can be imported only from the analysis cluster. To import data in CSV format stored on HDFS from MRS, select an existing MRS cluster and select the file name or directory from the HDFS file list. The number of columns in the imported file must be the same as that of the dataset schema.

### 2.4.3 Specifications for Importing the Manifest File

The manifest file defines the mapping between labeling objects and content. The **Manifest file** import mode means that the manifest file is used for dataset import. The manifest file can be imported from OBS. When importing a manifest file from OBS, ensure that the current user has the permissions to access the directory housing the manifest file.

**NOTE**

There are many requirements on the Manifest file compilation. You are advised to import new data from OBS. Generally, Manifest file import is used for data migration of ModelArts in different regions or using different accounts. That is, if you have labeled data in a region using ModelArts, you can obtain the manifest file of the published dataset from the output path. Then you can import the dataset using the manifest file to ModelArts of other regions or accounts. The imported data carries the labeling information and does not need to be labeled again, improving development efficiency.

The manifest file that contains information about the original file and labeling can be used in labeling, training, and inference scenarios. The manifest file that contains only information about the original file can be used in inference scenarios or used to generate an unlabeled dataset. The manifest file must meet the following requirements:

- The manifest file uses the UTF-8 encoding format. The **source** value of text classification can contain Chinese characters. However, Chinese characters are not recommended for other parameters.
ModelArts
User Guide (Senior AI Engineers)

●

2 Data Management

The manifest file uses the JSON Lines format (jsonlines.org). A line contains
one JSON object.

{"source": "/path/to/image1.jpg", "annotation": ... }
{"source": "/path/to/image2.jpg", "annotation": ... }
{"source": "/path/to/image3.jpg", "annotation": ... }

In the preceding example, the manifest file contains multiple lines of JSON
object.
The manifest file can be generated by users, third-party tools, or ModelArts
Data Labeling. The file name can be any valid file name. To facilitate the
internal use of the ModelArts system, the file name generated by the
ModelArts Data Labeling function consists of the following character strings:
DatasetName-VersionName.manifest. For example, animalv201901231130304123.manifest.

●

Image Classification
{

}

"source":"s3://path/to/image1.jpg",
"usage":"TRAIN",
"hard":"true",
"hard-coefficient":0.8,
"id":"0162005993f8065ef47eefb59d1e4970",
"annotation": [
{
"type": "modelarts/image_classification",
"name": "cat",
"property": {
"color":"white",
"kind":"Persian cat"
},
"hard":"true",
"hard-coefficient":0.8,
"annotated-by":"human",
"creation-time":"2019-01-23 11:30:30"
},
{
"type": "modelarts/image_classification",
"name":"animal",
"annotated-by":"modelarts/active-learning",
"confidence": 0.8,
"creation-time":"2019-01-23 11:30:30"
}],
"inference-loc":"/path/to/inference-output"

Table 2-11 Parameter description
Parameter

Mandator
y

Description

source

Yes

URI of an object to be labeled. For details about
data source types and examples, see Table 2-12.

Issue 01 (2021-01-15)

Copyright © Huawei Technologies Co., Ltd.

60


<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
</table>
| usage       | No        | By default, the parameter value is left blank. Possible values are as follows:  
  - **TRAIN**: The object is used for training.  
  - **EVAL**: The object is used for evaluation.  
  - **TEST**: The object is used for testing.  
  - **INFORMATION**: The object is used for inference.  
  If the parameter value is left blank, the user decides how to use the object. |
| id          | No        | Sample ID exported from the system. You do not need to set this parameter when importing the sample. |
| annotation  | No        | If the parameter value is left blank, the object is not labeled. The value of `annotation` consists of an object list. For details about the parameters, see Table 2-13. |
| inference-loc | No      | This parameter is available when the file is generated by the inference service, indicating the location of the inference result file. |

Table 2-12 Data source types

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBS</td>
<td>&quot;source&quot;:&quot;s3://path-to-jpg&quot;</td>
</tr>
<tr>
<td>Content</td>
<td>&quot;source&quot;:&quot;content://I love machine learning&quot;</td>
</tr>
</tbody>
</table>

Table 2-13 Description of `annotation` objects

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
</table>
| type      | Yes       | Label type. Possible values are as follows:  
  - **image_classification**: image classification  
  - **text_classification**: text classification  
  - **text_entity**: named entity recognition  
  - **object_detection**: object detection  
  - **audio_classification**: sound classification  
  - **audio_content**: speech labeling  
  - **audio_segmentation**: speech paragraph labeling |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Yes/No</td>
<td>This parameter is mandatory for the classification type but optional for other types. This example uses the image classification type.</td>
</tr>
<tr>
<td>id</td>
<td>Yes/No</td>
<td>Label ID. This parameter is mandatory for triplets but optional for other types. The entity label ID of a triplet is in E+number format, for example, E1 and E2. The relationship label ID of a triplet is in R+number format, for example, R1 and R2.</td>
</tr>
<tr>
<td>property</td>
<td>No</td>
<td>Labeling property. In this example, the cat has two properties: color and kind.</td>
</tr>
<tr>
<td>hard</td>
<td>No</td>
<td>Indicates whether the example is a hard example. True indicates that the labeling example is a hard example, and False indicates that the labeling example is not a hard example.</td>
</tr>
<tr>
<td>annotated-by</td>
<td>No</td>
<td>The default value is human, indicating manual labeling.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● human</td>
</tr>
<tr>
<td>creation-time</td>
<td>No</td>
<td>Time when the labeling job was created. It is the time when labeling information was written, not the time when the manifest file was generated.</td>
</tr>
<tr>
<td>confidence</td>
<td>No</td>
<td>Confidence score of machine labeling. The value ranges from 0 to 1.</td>
</tr>
</tbody>
</table>

**Text Classification**

```json
{
    "source": "content://I like this product ",
    "id": "XGDVGS",
    "annotation": [
    {
        "type": "modelarts/text_classification",
        "name": "positive",
        "annotated-by": "human",
        "creation-time": "2019-01-23 11:30:30"
    }
    ]
}
```

The `content` parameter indicates the text to be labeled (in UTF-8 encoding format, which can be Chinese). The other parameters are the same as those described in Image Classification. For details, see Table 2-11.

**Named Entity Recognition**

```json
{
    "source": "content://Michael Jordan is the most famous basketball player in the world.",
    "usage": "TRAIN",
    "annotation": [
    {
        "type": "modelarts/text_entity",
    }
    ]
}
```
The parameters such as source, usage, and annotation are the same as those described in Image Classification. For details, see Table 2-11.

**Table 2-14** describes the property parameters. For example, if you want to extract Michael from "source":"content://Michael Jordan", the value of start_index is 0 and that of end_index is 7.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@modelarts:start_index</td>
<td>Integer</td>
<td>Start position of the text. The value starts from 0, including the characters specified by start_index.</td>
</tr>
<tr>
<td>@modelarts:end_index</td>
<td>Integer</td>
<td>End position of the text, excluding the characters specified by end_index.</td>
</tr>
</tbody>
</table>

### Text Triplet

```
"source":"content://"Three Body" is a series of long science fiction novels created by Liu Cixin.",
"usage":"TRAIN",
"annotation":{
    
    "type":"modelarts/text_entity",
    "name":"Person",
    "id":"E1",
    "property":{
        "@modelarts:start_index":67,
        "@modelarts:end_index":74
    },
    "annotated-by":"human",
    "creation-time":"2019-01-23 11:30:30"
},

{ "type":"modelarts/text_entity",
  "name":"Book",
  "id":"E2",
  "property":{
      "@modelarts:start_index":0,
  
```
The parameters such as source, usage, and annotation are the same as those described in Image Classification. For details, see Table 2-11.

Table 5 property parameters describes the property parameters. @modelarts:start_index and @modelarts:end_index are the same as those of named entity recognition. For example, when source is set to content://"Three Body" is a series of long science fiction novels created by Liu Cix., Liu Cix is an entity person, Three Body is an entity book, the person is the author of the book, and the book is works of the person.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@modelarts:start_index</td>
<td>Integer</td>
<td>Start position of the triplet entities. The value starts from 0, including the characters specified by start_index.</td>
</tr>
<tr>
<td>@modelarts:end_index</td>
<td>Integer</td>
<td>End position of the triplet entities, excluding the characters specified by end_index.</td>
</tr>
<tr>
<td>@modelarts:from</td>
<td>String</td>
<td>Start entity ID of the triplet relationship.</td>
</tr>
<tr>
<td>@modelarts:to</td>
<td>String</td>
<td>Entity ID pointed to in the triplet relationship.</td>
</tr>
</tbody>
</table>

Object Detection


The parameters such as `source`, `usage`, and `annotation` are the same as those described in Image Classification. For details, see Table 2-11.

- The `annotation-loc` indicates the path for saving the label file. This parameter is mandatory for object detection but optional for other types.
- The `annotation-format` indicates the format of the label file. This parameter is optional. The default value is PASCAL VOC. Currently, only PASCAL VOC is supported.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folder</td>
<td>Yes</td>
<td>Directory where the data source is located</td>
</tr>
<tr>
<td>filename</td>
<td>Yes</td>
<td>Name of the file to be labeled</td>
</tr>
<tr>
<td>size</td>
<td>Yes</td>
<td>Image pixel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>width</strong>: image width. This parameter is mandatory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>height</strong>: image height. This parameter is mandatory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>depth</strong>: number of image channels. This parameter is mandatory.</td>
</tr>
<tr>
<td>segmented</td>
<td>Yes</td>
<td>Segmented or not</td>
</tr>
<tr>
<td>Parameter</td>
<td>Mandatory</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| object    | Yes       | Object detection information. Multiple `object{}` functions are generated for multiple objects.  
- **name**: class of the labeled content. This parameter is mandatory.  
- **pose**: shooting angle of the labeled content. This parameter is mandatory.  
- **truncated**: whether the labeled content is truncated (0 indicates that the content is not truncated). This parameter is mandatory.  
- **occluded**: whether the labeled content is occluded (0 indicates that the content is not occluded). This parameter is mandatory.  
- **difficult**: whether the labeled object is difficult to identify (0 indicates that the object is easy to identify). This parameter is mandatory.  
- **confidence**: confidence score of the labeled object. The value range is 0 to 1. This parameter is optional.  
- **bndbox**: bounding box type. This parameter is mandatory. For details about the possible values, see Table 2-17. |

**Table 2-17 Description of bounding box types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Shape</th>
<th>Labeling Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>point</td>
<td>Point</td>
<td>Coordinates of a point</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;x&gt;100&lt;y&gt;</code></td>
</tr>
<tr>
<td>line</td>
<td>Line</td>
<td>Coordinates of points</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;x1&gt;100&lt;y1&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;y1&gt;100&lt;x1&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;x2&gt;200&lt;y2&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;y2&gt;200&lt;x2&gt;</code></td>
</tr>
<tr>
<td>bndbox</td>
<td>Rectangle</td>
<td>Coordinates of the upper left and lower right points</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;xmin&gt;100&lt;xmin&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;ymin&gt;100&lt;ymin&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;xmax&gt;200&lt;xmax&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&lt;ymax&gt;200&lt;ymax&gt;</code></td>
</tr>
<tr>
<td>Type</td>
<td>Shape</td>
<td>Labeling Information</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>polygon</td>
<td>Polygon</td>
<td>Coordinates of points&lt;br&gt;&lt;x1&gt;100&lt;br&gt;&lt;y1&gt;100&lt;br&gt;&lt;x2&gt;200&lt;br&gt;&lt;y2&gt;100&lt;br&gt;&lt;x3&gt;250&lt;br&gt;&lt;y3&gt;150&lt;br&gt;&lt;x4&gt;200&lt;br&gt;&lt;y4&gt;200&lt;br&gt;&lt;x5&gt;100&lt;br&gt;&lt;y5&gt;200&lt;br&gt;&lt;x6&gt;50&lt;br&gt;&lt;y6&gt;150</td>
</tr>
<tr>
<td>circle</td>
<td>Circle</td>
<td>Center coordinates and radius&lt;br&gt;&lt;cx&gt;100&lt;br&gt;&lt;cy&gt;100&lt;br&gt;&lt;r&gt;50</td>
</tr>
</tbody>
</table>

Example:

```xml
<annotation>
    <folder>test_data</folder>
    <filename>260730932.jpg</filename>
    <size>
        <width>767</width>
        <height>959</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>point</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <point>
            <x1>456</x1>
            <y1>596</y1>
        </point>
    </object>
    <object>
        <name>line</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <line>
            <x1>133</x1>
            <y1>651</y1>
            <x2>229</x2>
            <y2>561</y2>
        </line>
    </object>
</annotation>
```
The parameters such as source, usage, and annotation are the same as those described in Image Classification. For details, see Table 2-11.

Speech Labeling

{  "source":"s3://path/to/audio1.wav",  "annotation":[]}

The parameters such as `source`, `usage`, and `annotation` are the same as those described in *Image Classification*. For details, see Table 2-11.

The `@modelarts:content` parameter in `property` indicates speech labeling. The data type is *String*.

### Speech Paragraph Labeling

```json
{
  "source": "s3://path/to/audio1.wav",
  "usage": "TRAIN",
  "annotation": [
    {
      "type": "modelarts/audio_segmentation",
      "property": {
        "@modelarts:start_time": "00:01:10.123",
        "@modelarts:end_time": "00:01:15.456",
        "@modelarts:source": "Tom",
        "@modelarts:content": "How are you?"
      },
      "annotated-by": "human",
      "creation-time": "2019-01-23 11:30:30"
    },
    {
      "type": "modelarts/audio_segmentation",
      "property": {
        "@modelarts:start_time": "00:01:22.754",
        "@modelarts:end_time": "00:01:24.145",
        "@modelarts:source": "Jerry",
        "@modelarts:content": "I'm fine, thank you."
      },
      "annotated-by": "human",
      "creation-time": "2019-01-23 11:30:30"
    }
  ]
}
```

The parameters such as `source`, `usage`, and `annotation` are the same as those described in *Image Classification*. For details, see Table 2-11.

Table 2-18 describes the `property` parameters.

### Table 2-18 Description of `property` parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@modelarts:start_time</td>
<td>String</td>
<td>Start time of the sound. The format is <code>hh:mm:ss.SSS</code>. <code>hh</code> indicates the hour, <code>mm</code> indicates the minute, <code>ss</code> indicates the second, and <code>SSS</code> indicates the millisecond.</td>
</tr>
</tbody>
</table>
### Parameter, Data Type, Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@modelarts:end_time</td>
<td>String</td>
<td>End time of the sound. The format is <code>hh:mm:ss.SSS</code>. hh indicates the hour, mm indicates the minute, ss indicates the second, and SSS indicates the millisecond.</td>
</tr>
<tr>
<td>@modelarts:source</td>
<td>String</td>
<td>Sound source</td>
</tr>
<tr>
<td>@modelarts:content</td>
<td>String</td>
<td>Sound content</td>
</tr>
</tbody>
</table>

### 2.5 Exporting Data

A dataset includes labeled and unlabeled data. You can select images or filter data based on the filter criteria and export to a new dataset or the specified OBS directory. In addition, you can view the task history to learn about the export records.

**NOTE**

Currently, only datasets of image classification, object detection, and free format types can be exported.
- For image classification datasets, only the label files in TXT format can be exported.
- For object detection datasets, only XML label files in Pascal VOC format can be exported.
- For free format datasets, all files of the datasets can be exported.

#### Exporting Data to a New Dataset

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.
2. In the dataset list, select the dataset of the object detection or image classification type and click the dataset name to go to the **Dashboard** tab page of the dataset.

**NOTE**

For a dataset of the free format type, you can click the dataset name to directly access the dataset details page and go to 4.

3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed.
4. On the dataset details page, select or filter data to be exported. Click **Export To** and choose **New Dataset** from the drop-down list.
Figure 2-47 Selecting or filtering images to be exported

5. In the displayed Export to New Dataset dialog box, enter the related information and click OK.

- **Name**: name of the new dataset
- **Storage Path**: input path of the new dataset, that is, the OBS path where the data to be exported is stored
- **Output Path**: output path of the new dataset, that is, the output path after labeling is complete. The output path cannot be the same as the storage path, and the output path cannot be a subdirectory of the storage path.
- **Export Content**: The options are Export the selected samples and Export all samples meeting filtering criteria.
- **Hard Example Filtering**: Select whether to enable hard example filtering.

Figure 2-48 Exporting to a new dataset

6. After the data is exported, you can view the new dataset in the dataset list.

**Exporting Data to OBS**

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.

2. In the dataset list, select the dataset of the object detection or image classification type and click the dataset name to go to the Dashboard tab page of the dataset.

   - **NOTE**
     
     For a dataset of the free format type, you can click the dataset name to directly access the dataset details page and go to 4.

3. On the Dashboard page of the dataset, click Label in the upper right corner. The dataset details page is displayed.

4. On the dataset details page, select or filter data to be exported. Click Export To and choose OBS from the drop-down list.
5. In the displayed **Export to OBS** dialog box, enter the related information and click **OK**.

**Storage Path**: path where the data to be exported is stored. You are advised not to save data to the input or output path of the current dataset.

**Export Content**: The options are **Export the selected samples** and **Export all samples meeting filtering criteria**.

**Hard Example**: Select whether to enable hard examples.

6. After the data is exported, you can view it in the specified path.

**Viewing the Task History**

When you export data to a new dataset or OBS, you can view the export task details in the **View Task History** dialog box.

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.

2. In the dataset list, select the dataset of the object detection or image classification type and click the dataset name to go to the **Dashboard** tab page of the dataset.

   **NOTE**

   For a dataset of the free format type, you can click the dataset name to directly access the dataset details page and go to **4**.

3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed.

4. On the dataset details page, select or filter data to be exported. Click **Export To** and choose **View Task History** from the drop-down list.

5. In the **View Task History** dialog box, view the export task history of the current dataset. Information about **Task ID**, **Created**, **Type**, **Path**, **Total**, and **Status** is included.
2.6 Modifying a Dataset

For a created dataset, you can modify its basic information to match service changes.

Prerequisites

You have created a dataset.

Modifying the Basic Information About a Dataset

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.
2. In the dataset list, choose More > Modify in the Operation column. Alternatively, you can click the dataset name to go to the Dashboard tab page of the dataset, and click Modify in the upper right corner.
3. Modify basic information about the dataset and then click OK. Refer to Table 2-19 for details.

**NOTE**

- For datasets of the Object detection type, team labeling cannot be disabled after being enabled.
- For object detection, labels with label attributes cannot be modified but can be deleted.

Figure 2-52 Modifying a dataset

Modify Dataset

<table>
<thead>
<tr>
<th>Name</th>
<th>dataset-yunbao-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>

Label Set

<table>
<thead>
<tr>
<th>yunbao</th>
<th>▼</th>
<th>+</th>
<th>● Add Label</th>
</tr>
</thead>
</table>

OK  Cancel
### Table 2-19 Parameter description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter the name of the dataset. A dataset name can contain only letters, digits, underscores (&quot;_&quot;), and hyphens (&quot;-&quot;).</td>
</tr>
<tr>
<td>Description</td>
<td>Enter a brief description for the dataset.</td>
</tr>
<tr>
<td>Label Set</td>
<td>The label set varies depending on the dataset type. For details about how to modify the label set, see the parameter description of different dataset types in <em>Creating a Dataset</em>.</td>
</tr>
</tbody>
</table>

## 2.7 Publishing a Dataset

ModelArts distinguishes data of the same source according to versions labeled at different time, which facilitates the selection of dataset versions during subsequent model building and development. After labeling the data, you can publish the dataset to generate a new dataset version.

### About Dataset Versions

- For a newly created dataset (before publishing), there is no dataset version information. The dataset must be published before being used for model development or training.
- The default naming rules of dataset versions are V001 and V002 in ascending order. You can customize the version number during publishing.
- You can set any version to the current directory. Then the details of the version are displayed on the dataset details page.
- You can obtain the dataset in the manifest file format corresponding to each dataset version based on the value of *Storage Path*. The dataset can be used when you import data or filter hard examples.
- The version of a table dataset cannot be changed.

### Publishing a Dataset

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.
2. In the dataset list, click **Publish** in the **Operation** column. Alternatively, you can click **Publish** in the **Dashboard** tab page of the dataset, and click **Publish** in the upper right corner.
3. In the displayed dialog box, set the parameters and click **OK**.
### Table 2-20 Parameters for publishing a dataset

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version Name</td>
<td>The naming rules of V001 and V002 in ascending order are used by default. A version name can be customized. Only letters, digits, hyphens (-), and underscores (_) are allowed.</td>
</tr>
<tr>
<td>Format</td>
<td>Available values are <strong>Default</strong> and <strong>CarbonData</strong>.</td>
</tr>
<tr>
<td>Splitting</td>
<td>By default, this function is disabled. After this function is enabled, you need to set the training and validation ratios. Enter a value ranging from 0 to 1 for <strong>Training Set Ratio</strong>. After the training set ratio is set, the validation set ratio is determined. The sum of the training set ratio and the validation set ratio is 1. The training set ratio is the ratio of sample data used for model training. The validation set ratio is the ratio of the sample data used for model validation. The training and validation ratios affect the performance of training templates.</td>
</tr>
<tr>
<td>Description</td>
<td>Description of the current dataset version.</td>
</tr>
<tr>
<td>Hard Example</td>
<td>By default, this function is disabled. After this function is enabled, information such as the hard example attributes of the dataset are written to the corresponding manifest file.</td>
</tr>
</tbody>
</table>

### Figure 2-53 Publishing a dataset

![Publish New Version](image)

After the version is published, you can go to the **Version Manager** tab page to view the detailed information. By default, the system sets the latest version to the current directory.

### Directory Structure of Related Files After the Dataset Is Published

Datasets are managed based on OBS directories. After a new version is published, the directory is generated based on the new version in the output dataset path.
Take an image classification dataset as an example. After the dataset is published, the directory structure of related files generated in OBS is as follows:

|-- user-specified-output-path
  |-- DatasetName-datasetId
    |-- annotation
      |-- VersionMame1
        |-- VersionMame1.manifest
        |-- VersionMame2
      |-- ...

The following uses object detection as an example. If a manifest file is imported to the dataset, the following provides the directory structure of related files after the dataset is published:

|-- user-specified-output-path
  |-- DatasetName-datasetId
    |-- annotation
      |-- VersionMame1
        |-- VersionMame1.manifest
        |-- annotation
        |-- file1.xml
      |-- VersionMame2
        ...

2.8 Deleting a Dataset

If a dataset is no longer in use, you can delete it to release resources.

NOTE

After a dataset is deleted, if you need to delete the data in the dataset input and output paths in OBS to release resources, you are advised to delete the data and the OBS folders on the OBS Console.

Procedure

1. In the left navigation pane, choose Data Management > Datasets. On the Datasets page, choose More > Delete in the Operation column of the dataset.

2. In the displayed dialog box, click OK.

NOTE

After a dataset is deleted, some functions such as dataset version management become unavailable. Exercise caution when performing this operation. However, the original data and labeling data of the dataset are still stored in OBS.

2.9 Managing Dataset Versions

After labeling data, you can publish the dataset to multiple versions for management. For the published versions, you can view the dataset version updates, set the current version, and delete versions. For details about dataset versions, see About Dataset Versions.

For details about how to publish a new version, see Publishing a Dataset.
Viewing Dataset Version Updates

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.
2. In the dataset list, choose More > Manage Version in the Operation column. The Manage Version tab page is displayed.

You can view basic information about the dataset, and view the versions and publish time on the left.

![Figure 2-54 Viewing dataset versions](image1)

Setting to Current Version

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.
2. In the dataset list, choose More > Manage Version in the Operation column. The Manage Version tab page is displayed.
3. On the Manage Version tab page, select the desired dataset version, and click Set to Current Version in the basic information area on the right. After the setting is complete, Current version is displayed to the right of the version name.

**NOTE**

Only the version in Normal status can be set to the current version.

![Figure 2-55 Setting to current version](image2)
Deleting a Dataset Version

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.
2. In the dataset list, choose More > Manage Version in the Operation column. The Manage Version tab page is displayed.
3. Locate the row that contains the target version, and click Delete in the Operation column. In the dialog box that is displayed, click OK.

**NOTE**
Deleting a dataset version does not remove the original data. Data and its labeling information are still stored in the OBS directory. However, if it is deleted, you cannot manage the dataset versions on the ModelArts management console. Exercise caution when performing this operation.

2.10 Auto Labeling

In addition to manual labeling, ModelArts also provides the auto labeling function to quickly label data, reducing the labeling time by more than 70%. Auto labeling means learning and training are performed based on the selected labels and images and an existing model is selected to quickly label the remaining images.

**Background**
- Currently, only datasets of image classification and object detection types support the auto labeling function.
- To enable Auto Labeling, add at least two types of labels to the dataset and add each type of the label to at least 5 objects.
- At least one unlabeled image must exist when you enable Auto Labeling.
- Before enabling Auto Labeling, ensure that no auto labeling task is in progress in the system.
- Check the image data used for labeling and ensure that no RGBA four-channel image exists in the image data. If four-channel images exist, the auto labeling task will fail. Therefore, delete the four-channel images from the dataset and then start the auto labeling task.

**Auto Labeling**

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.
2. In the dataset list, select a dataset of the object detection or image classification type and click Auto Labeling in the Operation column to start an intelligent labeling job.
3. On the Enable Auto Labeling page, select Active learning or Pre-labeling. For details, see Table 2-21 and Table 2-22.
Table 2-21 Active learning

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Labeling Type</td>
<td><strong>Active learning</strong>: The system uses semi-supervised learning and hard example filtering to perform auto labeling, reducing manual labeling workload and helping you find hard examples.</td>
</tr>
<tr>
<td>Algorithm Type</td>
<td>For a dataset of the image classification type, you need to set the following parameters:</td>
</tr>
<tr>
<td></td>
<td><strong>Fast</strong>: Use the labeled samples for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Precise</strong>: Use labeled and unlabeled samples for semi-supervised training, which improves the model precision.</td>
</tr>
</tbody>
</table>

Table 2-22 Pre-labeling

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Labeling Type</td>
<td><strong>Pre-labeling</strong>: Select a model displayed on the Model Management page. Ensure that the model type matches the dataset labeling type. After the pre-labeling is complete, if the labeling result complies with the standard labeling format defined by the platform, the system filters hard examples. This step does not affect the pre-labeling result.</td>
</tr>
<tr>
<td>Model and Version</td>
<td>● <strong>My Models</strong>: You can select a model based on site requirements. Click the drop-down arrow on the left of the target model and select a proper version. For details about how to import a model, see Importing a Model.</td>
</tr>
<tr>
<td>Specifications</td>
<td>In the drop-down list, you can select the node specifications supported by ModelArts.</td>
</tr>
<tr>
<td>Compute Nodes</td>
<td>The default value is 1. You can select a value based on site requirements. The maximum value is 5.</td>
</tr>
</tbody>
</table>

**NOTE**

- For datasets of the object detection type, only rectangular boxes can be recognized and labeled when **Active Learning** is selected.
- If there are too many auto labeling jobs in the system, the jobs may be queued. As a result, the jobs are always in the labeling state. The system will complete labeling jobs in sequence.
**Figure 2-56** Enabling auto labeling (image classification)

**Enable Auto Labeling**

Auto Labeling Type
- Active learning
- Pre-labeling

The system uses semi-supervised learning and hard example filtering to perform auto labeling, reducing manual labeling workload and helping you find hard examples.

⚠️ Auto Labeling can identify and add only rectangle labeling boxes.

**Limited-time free**
Auto Labeling is billed based on the training duration. Pricing details

**Figure 2-57** Enabling auto labeling (object detection)

**Enable Auto Labeling**

Auto Labeling Type
- Active learning
- Pre-labeling

The system uses semi-supervised learning and hard example filtering to perform auto labeling, reducing manual labeling workload and helping you find hard examples.

⚠️ Auto Labeling can identify and add only rectangle labeling boxes.

**Limited-time free**
Auto Labeling is billed based on the training duration. Pricing details

**Figure 2-58** Enabling auto labeling (pre-labeling)

**Enable Auto Labeling**

Auto Labeling Type
- Active learning
- Pre-labeling

Select a model created on the Model Management page. Ensure that the model type matches the dataset labeling type. After the pre-labeling is complete, if the labeling result complies with the standard labeling format defined by the platform, the system filters hard examples. This step does not affect the pre-labeling result.

- Model and Version
- Specifications
- Compute Nodes

**Limited-time free**
Auto Labeling is billed based on the training duration. Pricing details
4. After setting the parameters, click **Submit** to enable auto labeling.
5. In the dataset list, click a dataset name to go to the **Dashboard** page.
6. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed.
7. On the dataset details page, click the **To be Confirmed** tab to view the auto labeling progress.
   You can also enable auto labeling or view the auto labeling history on this tab page.

   **Figure 2-59** Labeling progress

8. After auto labeling is complete, all the labeled images are displayed on the **To Be Confirmed** page.
   - **Datasets of the image classification type**
     On the **To Be Confirmed** page, check whether labels are correct, select the correctly labeled images, and click **Labeled** to confirm the auto labeling results. The confirmed image will be categorized to the **Labeled** page.
     You can manually modify the labels of the images marked as hard examples based on site requirements. For details, see **For datasets of the image classification type**.
   - **Datasets of the object detection type**
     On the **To Be Confirmed** page, click images to view their labeling details and check whether labels and target bounding boxes are correct. For the correctly labeled images, click **Labeled** to confirm the auto labeling results. The confirmed image will be categorized to the **Labeled** page.
     You can manually modify the labels or target bounding boxes of the images marked as hard examples based on site requirements. For details, see **For datasets of the object detection type**.

### 2.11 Confirming Hard Examples

In a labeling task that processes a large amount of data, auto labeling results cannot be directly used for training because the labeled images are insufficient at the initial stage of labeling. It takes a lot of time and manpower to adjust and confirm all unlabeled data one by one. To accelerate labeling progress, ModelArts embeds an auto hard example detection function for labeling unlabeled data in an
auto labeling task. This function provides suggestions on labeling priorities for remaining unlabeled images. The auto labeling result of an image with high labeling priority is not as expected. Therefore, this case is called a hard example.

The auto hard example detection function is used to automatically label hard examples during auto labeling and data collection and filtering. You are advised to further confirm and label hard example data, and add labeling results to the training dataset to obtain a trained model with higher precision. No manual intervention is required for hard example detection, and you only need to confirm and modify the labeled data, improving data management and labeling efficiency. In addition, you can supplement data similar to hard examples to improve the variety of the dataset and further improve the model training precision.

Hard example management involves three scenarios. See Figure 2-60.

- **Confirming Hard Examples After Auto Labeling**
- **Labeling Data in a Dataset as Hard Examples**
- **Viewing the Collected Hard Examples**

![Figure 2-60 Hard examples in a dataset](image)

**NOTE**

Currently, only datasets of image classification and object detection types support the auto hard example detection function.

**Confirming Hard Examples After Auto Labeling**

During the execution of an auto labeling task, ModelArts automatically detects and labels hard examples. After the auto labeling task is complete, the labeling results of hard examples are displayed on the **To Be Confirmed** tab page. You are advised to modify hard example data and confirm the labeling result.

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.
2. In the dataset list, select the dataset of the object detection or image classification type and click the dataset name to go to the **Dashboard** tab page of the dataset.
3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed.
4. On the dataset details page, click the **To Be Confirmed** tab to view and confirm hard examples.

**NOTE**

Labeling data is displayed on the **To Be Confirmed** tab page only after the auto labeling task is complete. Otherwise, no data is available on the tab page. For details about auto labeling, see **Auto Labeling**.

- For datasets of the object detection type

On the **To Be Confirmed** tab page, click an image to expand its labeling details. Check whether labeling information is correct, for example, whether the label is correct and whether the target bounding box is correctly added to the right position. If the auto labeling result is inaccurate, you are advised to manually adjust the label or target bounding box and click **Labeled**. Then, the re-labeled data is displayed on the **Labeled** tab page.

In the hard example shown in **Figure 2-61**, the position of the target bounding box for the **dog** label is incorrect. Use a bounding box to re-label the image again, for example, the **miss** bounding box shown in the following figure. Then, delete the incorrect labeling bounding box, that is, the **false** bounding box. After manual adjustment, click **Labeled** to confirm the hard example.

**Figure 2-61** Confirming a hard example for object detection

- For datasets of the image classification type

On the **To Be Confirmed** tab page, check whether labels added to images with the **Hard example** mark are correct. Select the images that are incorrectly labeled, delete the incorrect labels, and add correct labels in **Label** on the right. Click **OK**. The selected images and its labeling details are displayed on the **Labeled** tab page.

As shown in **Figure 2-62**, the selected images are incorrectly labeled. Delete the incorrect labels on the right, add the **dog** label in **Label**, and click **OK** to confirm the hard examples.
Labeling Data in a Dataset as Hard Examples

In a dataset, labeled or unlabeled image data can be labeled as hard examples. Data labeled as hard examples can be used to improve model precision through built-in rules during subsequent model training.

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.
2. In the dataset list, select the dataset of the object detection or image classification type and click the dataset name to go to the **Dashboard** tab page of the dataset.
3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed.
4. On the dataset details page, click the **Labeled**, **Unlabeled**, or **All** tab, select the images to be labeled as hard examples, and choose **Batch process Hard Examples > Confirm Hard Example**. After the labeling is complete, a **Hard example** mark will be displayed in the upper right corner of a preview image.

Viewing the Collected Hard Examples

For real-time or batch services, you can create data collection and filtering tasks. Based on the rules you set, the system automatically detects hard examples in the collected data and stores them in the corresponding datasets. For details about the configuration description and operations, see **Collecting Data** (real-time service) and **Collecting Data** (batch service).
1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.

2. In the dataset list, select a dataset specified in the data collection task, and click the dataset name to go to the **Dashboard** tab page of the dataset.

3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed.

4. On the dataset details page, click the **To Be Confirmed** tab to view and confirm hard examples.
   - You can manually modify labeling information about data that has been labeled as hard examples and confirm the labeling result. For details about how to confirm hard examples for the two types of datasets, refer to 4.
   - You can add data similar to hard examples to the dataset so that model precision can be improved during the next model training.

### 2.12 Auto Grouping

To improve the precision of auto labeling algorithms, you can evenly label multiple classes. ModelArts provides built-in grouping algorithms. You can enable auto grouping to improve data labeling efficiency.

Auto grouping can be understood as data labeling preprocessing. Clustering algorithms are used to cluster unlabeled images, and images are labeled or cleaned by group based on the clustering result.

For example, a user searches for `XX` through a search engine, downloads and uploads related images to the dataset, and then uses the auto grouping function to classify `XX` images, such as papers, posters, images confirmed as `XX`, and others. The user can quickly remove unwanted images from a group or select all images of a type and add labels to the images.

**NOTE**

Currently, only datasets of image classification and object detection types support the auto grouping function.

### Starting Auto Grouping Tasks

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.

2. In the dataset list, select the dataset of the object detection or image classification type and click the dataset name to go to the **Dashboard** tab page of the dataset.

3. On the **Dashboard** page of the dataset, click **Label** in the upper right corner. The dataset details page is displayed.

4. On the **All** tab page of the dataset details page, choose **Auto Grouping > Start Task**.

**NOTE**

You can start auto group tasks or view task history only on the **All** tab page.
5. In the displayed **Auto Grouping** dialog box, set parameters and click **OK**.
   - **Groups**: Enter an integer from 2 to 200. The parameter value indicates the number of groups into which images are divided.
   - **Result Processing Method**: Select **Update attribute** or **Save to OBS**.
   - **Attribute Name**: If you select **Update attribute**, you need to enter an attribute name.
   - **Result Storage Path**: If you select **Save to OBS**, specify an OBS path.
   - **Advanced Feature Settings**: After this function is enabled, you can select **Clarity**, **Brightness**, and **Color** dimensions for the auto grouping function so that the grouping is based on the image brightness, color, and clarity. You can select multiple options.

![Auto grouping](image)

**Figure 2-64** Auto grouping

6. After the task is submitted, the task progress is displayed in the upper right corner of the page. After the task is complete, you can view the history of the auto grouping tasks to learn task status.

**Viewing the Auto Grouping Result**

On the **All** tab page of the dataset details page, expand **Filter Criteria**, set **Sample Attribute** to the attribute name of the auto grouping task, and set the sample attribute value to filter the grouping result.

![Viewing the auto grouping result](image)

**Figure 2-65** Viewing the auto grouping result
Viewing Auto Grouping Task History

On the All tab page of the dataset details page, choose Auto Grouping > View Task History. In the View Task History dialog box, basic information about the auto grouping tasks of the current dataset is displayed.

**Figure 2-66 Auto grouping task history**

View Task History

<table>
<thead>
<tr>
<th>Created</th>
<th>Groups</th>
<th>Result Processing</th>
<th>Storage Path/Attr.</th>
<th>Status</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-03-13 09:20...</td>
<td>2</td>
<td>Update attribute</td>
<td>sunflowers</td>
<td>Running</td>
<td>The j...</td>
</tr>
</tbody>
</table>

2.13 Data Features

Images or target bounding boxes are analyzed based on image features, such as blurs and brightness to draw visualized curves to help process datasets.

You can also select multiple versions of a dataset to view their curves for comparison and analysis.

**Background**

- Data feature analysis is only available for datasets whose labeling type is **Object detection** or **Image classification**.
- Data feature analysis is only available for the published datasets. The published dataset versions in Default format support data feature analysis.
- A data scope for feature analysis varies depending on the dataset type.
  - In a dataset of the object detection type, if the number of labeled samples is 0, the Data Features tab page is unavailable and data features are not displayed after a version is published. After the images are labeled and the version is published, and the data features of the labeled images are displayed.
  - In a dataset of the image classification type, if the number of labeled samples is 0, the Data Features tab page is unavailable and data features are not displayed after a version is published. After the images are labeled and the version is published, and the data features of all images are displayed.
- The analysis result is valid only when the number of images in a dataset reaches a certain level. Generally, more than 1,000 images are required.
- Image classification supports the following data feature metrics: Resolution, Aspect Ratio, Brightness, Saturation, Blur Score, and Colorfulness Object detection supports all data feature metrics. **Supported Data Feature Metrics** provides all data feature metrics supported by ModelArts.
Data Feature Analysis

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. The Datasets page is displayed.

2. Select a dataset and click Data Features in the Operation column. The Data Features tab page of the dataset page is displayed.

You can also click a dataset name to go to the dataset page and click the Data Features tab.

3. By default, feature analysis is not started for published datasets. You need to manually start feature analysis tasks for each dataset version. On the Data Features tab page, click Feature Analysis.

4. In the dialog box that is displayed, configure the dataset version for feature analysis and click OK to start analysis.

   Version: Select a published version of the dataset.

5. After a data feature analysis task is started, it takes a certain period of time to complete, depending on the data volume. If the selected version is displayed in the Version drop-down list and can be selected, the analysis is complete.

6. View the data feature analysis result.

   Version: Select the version to be compared from the drop-down list. You can also select only one version.

   Type: Select the type to be analyzed. The value can be all, train, eval, or inference.
**Data Feature Metric**: Select metrics to be displayed from the drop-down list. For details, see **Supported Data Feature Metrics**.

Then, the selected version and metrics are displayed on the page, as shown in **Figure 2-70**. The displayed chart helps you understand data distribution for better data processing.

**Figure 2-70 Data feature analysis**

7. View historical records of the analysis task.

After data feature analysis is complete, you can click **Task History** on the right of the **Data Features** tab page to view historical analysis tasks and their statuses in the dialog box that is displayed.

**Figure 2-71 Viewing the task history**

View Task History

<table>
<thead>
<tr>
<th>Dataset Vers.</th>
<th>Task ID</th>
<th>Created</th>
<th>Duration (hh:mm)</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>V002</td>
<td>Rdmjvwum33T</td>
<td>Apr 03, 2020</td>
<td>00:01:50</td>
<td>Successful</td>
</tr>
<tr>
<td>V001</td>
<td>gpw2hG6D6</td>
<td>Apr 02, 2020</td>
<td>00:02:23</td>
<td>Successful</td>
</tr>
</tbody>
</table>
## Supported Data Feature Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>Image resolution. An area value is used as a statistical value.</td>
<td>Metric analysis results are used to check whether there is an offset point. If an offset point exists, you can resize or delete the offset point.</td>
</tr>
<tr>
<td>Aspect Ratio</td>
<td>An aspect ratio is a proportional relationship between an image's width and height.</td>
<td>The chart of the metric is in normal distribution, which is generally used to compare the difference between the training set and the dataset used in the real scenario.</td>
</tr>
<tr>
<td>Brightness</td>
<td>Brightness is the perception elicited by the luminance of a visual target. A larger value indicates better image brightness.</td>
<td>The chart of the metric is in normal distribution. You can determine whether the brightness of the entire dataset is high or low based on the distribution center. You can adjust the brightness based on your application scenario. For example, if the application scenario is night, the brightness should be lower.</td>
</tr>
<tr>
<td>Saturation</td>
<td>Color saturation of an image. A larger value indicates that the entire image color is easier to distinguish.</td>
<td>The chart of the metric is in normal distribution, which is generally used to compare the difference between the training set and the dataset used in the real scenario.</td>
</tr>
<tr>
<td>Blur Score Clarity</td>
<td>Image clarity, which is calculated using the Laplace operator. A larger value indicates clearer edges and higher clarity.</td>
<td>You can determine whether the clarity meets the requirements based on the application scenario. For example, if data is collected from HD cameras, the clarity must be higher. You can sharpen or blur the dataset and add noises to adjust the clarity.</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Colorfulness</td>
<td>Horizontal coordinate: Colorfulness of an image. A larger value indicates richer colors. Vertical coordinate: Number of images</td>
<td>Colorfulness on the visual sense, which is generally used to compare the difference between the training set and the dataset used in the real scenario.</td>
</tr>
<tr>
<td>Bounding Box Number</td>
<td>Horizontal coordinate: Number of bounding boxes in an image Vertical coordinate: Number of images</td>
<td>It is difficult for a model to detect a large number of bounding boxes in an image. Therefore, more images containing many bounding boxes are required for training.</td>
</tr>
<tr>
<td>Std of Bounding Boxes Area Per Image</td>
<td>Horizontal coordinate: Standard deviation of bounding boxes in an image. If an image has only one bounding box, the standard deviation is 0. A larger standard deviation indicates higher bounding box size variation in an image. Vertical coordinate: Number of images</td>
<td>It is difficult for a model to detect a large number of bounding boxes with different sizes in an image. You can add data for training based on scenarios or delete data if such scenarios do not exist.</td>
</tr>
<tr>
<td>Aspect Ratio of Bounding Boxes</td>
<td>Horizontal coordinate: Aspect ratio of the target bounding boxes Vertical coordinate: Number of bounding boxes in all images</td>
<td>The chart of the metric is generally in Poisson distribution, which is closely related to application scenarios. This metric is mainly used to compare the differences between the training set and the validation set. For example, if the training set is a rectangle, the result will be significantly affected if the validation set is close to a square.</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Area Ratio of Bounding Boxes</td>
<td>Horizontal coordinate: Area ratio of the target bounding boxes, that is, the ratio of the bounding box area to the entire image area. A larger value indicates a higher ratio of the object in the image. Vertical coordinate: Number of bounding boxes in all images</td>
<td>The metric is used to determine the distribution of anchors used in the model. If the target bounding box is large, set the anchor to a large value.</td>
</tr>
<tr>
<td>Marginalization Value of Bounding Boxes</td>
<td>Horizontal coordinate: Marginalization degree, that is, the ratio of the distance between the center point of the target bounding box and the center point of the image to the total distance of the image. A larger value indicates that the object is closer to the edge. Vertical coordinate: Number of bounding boxes in all images</td>
<td>Generally, the chart of the metric is in normal distribution. The metric is used to determine whether an object is at the edge of an image. If a part of an object is at the edge of an image, you can add a dataset or do not label the object.</td>
</tr>
<tr>
<td>Overlap Score of Bounding Boxes</td>
<td>Horizontal coordinate: Overlap degree, that is, the part of a single bounding box overlapped by other bounding boxes. The value ranges from 0 to 1. A larger value indicates that more parts are overlapped by other bounding boxes. Vertical coordinate: Number of bounding boxes in all images</td>
<td>The metric is used to determine the overlapping degree of objects to be detected. Overlapped objects are difficult to detect. You can add a dataset or do not label some objects based on your needs.</td>
</tr>
<tr>
<td>Brightness of Bounding Boxes</td>
<td>Horizontal coordinate: Brightness of the image in the target bounding box. A larger value indicates brighter image. Vertical coordinate: Number of bounding boxes in all images</td>
<td>Generally, the chart of the metric is in normal distribution. The metric is used to determine the brightness of an object to be detected. In some special scenarios, the brightness of an object is low and may not meet the requirements.</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Blur Score of Bounding Boxes</td>
<td>Horizontal coordinate: Clarity of the image in the target bounding box. A larger value indicates higher image clarity. Vertical coordinate: Number of bounding boxes in all images</td>
<td>The metric is used to determine whether the object to be detected is blurred. For example, a moving object may become blurred during collection and its data needs to be collected again.</td>
</tr>
</tbody>
</table>

### 2.14 Deploying a Model in One Click

You can use the one-click model deployment function to create training jobs for the labeled dataset, import a model, and deploy the model as a real-time service all in one step. You can quickly deploy an available service without multiple steps.

**Background**

- The one-click model deployment function is only available for datasets whose labeling type is object detection and image classification.
- Currently, only the built-in algorithms can be used for training. You are advised to separately create a training job for a task that is trained using frequently-used frameworks or custom images.
- After a one-click model deployment task is created, the system automatically creates a training job, imports the trained model to ModelArts, and deploys the model as a real-time service. You only need to set parameters once to complete dataset-based AI development.
- Expense description:
  - Training jobs are billed in pay-per-use mode based on the resources you select. Training once is billed one time, and no extra fee is generated.
  - Currently, the imported models are not billed.
  - After the task is created, the created real-time service is in the **Running** status. If the public resource pool is used, the service is always being billed. Stop the service based on the site requirements to avoid unnecessary fees.
- The OBS directory you use and ModelArts are in the same region.

**Creating a Task**

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.
2. In the dataset list, choose **Deploy Model > Create Task**. The **Deploy Model** page is displayed.
3. Enter a task name and description.
4. Set parameters related to training. See **Figure 2-72**.

When you create a one-click model deployment task, the training parameters are similar to those of the training jobs. You can only select a built-in
algorithm for training. A built-in algorithm will be displayed based on the type of your dataset by default. For datasets of the object detection type, the available built-in algorithm is *Faster_RCNN_ResNet_v1_50*. For datasets of the image classification type, the available built-in algorithm is *ResNet_v1_50*. Running parameters vary depending on the built-in algorithm. Retain the default values for *Running Parameter*.

Set the following parameters: *Training Output Path, Job Log Path, Resource Pool, Type, Specifications*, and *Compute Nodes*. For details about how to set the parameters, see *Using Built-in Algorithms to Train Models*.

**Figure 2-72** Training information

5. Set deployment parameters. See **Figure 2-73**.

Select the resources used for real-time service deployment by setting the following parameters: *Resource Pool, Instance Flavor, Instance Count*, and *Environment Variable*. For details about how to set the parameters, go to 3 in *Deploying a Model as a Real-Time Service*.

**Figure 2-73** Parameters related to deployment
6. Set **Sample Collection**. This function is disabled by default. To enable this function, configure the related parameters. For details, see [Collecting Data](#).

7. After confirming the entered information, complete the task creation as prompted.

The task creation process includes the **Initialize**, **Train**, **Generate Model**, and **Deploy** steps. The time required varies according to dataset size.

After the **Train**, **Generate Model**, and **Deploy** steps are complete, the **View Training Details**, **View Model Details**, and **View Service Details** links are displayed on the page. You can click these links to view details.

![Figure 2-74 Task creation process](image)

### Viewing Task History

1. Log in to the ModelArts management console. In the left navigation pane, choose **Data Management > Datasets**. The **Datasets** page is displayed.

2. In the dataset list, choose **Deploy Model > View Historical Task**. The **One-Click Model Deployment Task History** page is displayed. **Figure 2-75** shows the details.

![Figure 2-75 Historical task details](image)

### 2.15 Team Labeling

#### 2.15.1 Team Labeling Overview

Generally, a small data labeling task can be completed by an individual. However, team work is required to label a large dataset. ModelArts provides the team labeling function. A labeling team can be formed to manage labeling for the same dataset.

**NOTE**

Currently, the team labeling function supports only datasets for image classification, object detection, text classification, named entity recognition, and text triplet.

### How to Enable Team Labeling

- When creating a dataset, enable **Team Labeling** and select a team or task manager.
Figure 2-76 Enabling during dataset creation

- If team labeling is not enabled for a dataset that has been created, create a team labeling task to enable team labeling. For details about how to create a team labeling task, see Creating Team Labeling Tasks.

Figure 2-77 Creating a team labeling task in a dataset list

Figure 2-78 Creating a team labeling task

Figure 2-79 Creating a team labeling task on the dataset details page
Operations Related to Team Labeling

- **Team Management**
- **Member Management**
- **Managing Team Labeling Tasks**

### 2.15.2 Team Management

Team labeling is managed in a unit of teams. To enable team labeling for a dataset, a team must be specified. Multiple members can be added to a team.

#### Background

- An account can have a maximum of 10 teams.
- An account must have at least one team to enable team labeling for datasets. If the account has no team, add a team by referring to [Adding a Team](#).

#### Adding a Team

1. In the left navigation pane of the ModelArts management console, choose **Data Management > Labeling Teams**. The **Labeling Teams** page is displayed.
2. On the **Labeling Teams** page, click **Add Team**.
3. In the displayed **Add Team** dialog box, enter a team name and description and click **OK**. The labeling team is added.

**Figure 2-80** Adding a team

<table>
<thead>
<tr>
<th>Add Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: team-6e6d</td>
</tr>
<tr>
<td>Description</td>
</tr>
</tbody>
</table>

The new team is displayed on the **Labeling Teams** page. You can view team details in the right pane. There is no member in the new team. Add members to the new team by referring to [Adding a Member](#).

#### Deleting a Team

You can delete a team that is no longer used.
On the **Labeling Teams** page, select the target team and click **Delete**. In the dialog box that is displayed, click **OK**.

**Figure 2-81** Deleting a team

![Labeling Teams](image)

### 2.15.3 Member Management

There is no member in a new team. You need to add members who will participate in a team labeling task.

A maximum of 100 members can be added to a team. If there are more than 100 members, you are advised to add them to different teams for better management.

**Adding a Member**

1. In the left navigation pane of the ModelArts management console, choose **Data Management > Labeling Teams**. The **Labeling Teams** page is displayed.
2. On the **Labeling Teams** page, select a team from the team list on the left and click a team name. The team details are displayed in the right pane.
3. In the **Team Details** area, click **Add Member**.
4. In the displayed **Add Member** dialog box, enter an email address, description, and a role for a member and click **OK**.

An email address uniquely identifies a team member. Different members cannot use the same email address. The email address you enter will be recorded and saved in ModelArts. It is used only for ModelArts team labeling. After a member is deleted, the email address will also be deleted.

Possible values of **Role** are **Labeler**, **Reviewer**, and **Team Manager**. Only one **Team Manager** can be set.
Figure 2-82 Adding a member

Information about the added member is displayed in the Team Details area.

Modifying Member Information

You can modify member information if it is changed.

1. In the Team Details area, select the desired member.
2. In the row containing the desired member, click Modify in the Operation column. In the displayed dialog box, modify the description or role.

Information about the added member is displayed in the Team Details area.

The email address of a member cannot be changed. To change the email address of a member, you are advised to delete the member, and set a new email address when adding a member.

Possible values of Role are Labeler, Reviewer, and Team Manager. Only one Team Manager can be set.

Deleting Members

- Deleting a single member
  In the Team Details area, select the desired member, and click Delete in the Operation column. In the dialog box that is displayed, click OK.

- Batch Deletion
  In the Team Details area, select members to be deleted and click Delete. In the dialog box that is displayed, click OK.
% 2.15.4 Managing Team Labeling Tasks

For datasets with team labeling enabled, you can create team labeling tasks and assign the labeling tasks to different teams so that team members can complete the labeling tasks together. During data labeling, members can initiate acceptance, continue acceptance, and view acceptance reports.

Creating Team Labeling Tasks

If you enable team labeling when creating a dataset and assign a team to label the dataset, the system creates a labeling task based on the team by default. After the dataset is created, you can view the labeling task on the Labeling Progress tab page of the dataset.

You can also create a team marking task and assign it to different members in the same team or to other labeling teams.

1. Log in to the ModelArts management console. In the left navigation pane, choose Data Management > Datasets. A dataset list is displayed.
2. In the dataset list, select a dataset that supports team labeling, and click the dataset name to go to the Dashboard tab page of the dataset.
3. Click the Labeling Progress tab to view existing labeling tasks of the dataset. Click Create Team Labeling Task in the upper right corner to create a task.

4. In the displayed Create Team Labeling Task dialog box, set related parameters and click OK.
   - Name: Enter a task name.
   - Type: Select a task type, Team or Task Manager.
   - Select Team: If Type is set to Team, you need to select a team and members for labeling. The Select Team drop-down list lists the labeling teams and members created by the current account. For details about team management, see Team Labeling Overview.
   - Select Task Manager: If Type is set to Task Manager, you need to select one Team Manager member from all teams as the task manager.
- **Label Set**: All existing labels and label attributes of the dataset are displayed. You can also select **Automatically synchronize new images to the team labeling task** or **Automatically load the intelligent labeling results to images that need to be labeled** under **Label Set**.

**Figure 2-85** Creating a team labeling task

After the task is created, you can view the new task on the **Labeling Progress** tab page.

**Labeling (Team Member)**

After a labeling task is created, the team member to which the task is assigned receives a labeling notification email.

In the email details, click the labeling task link and use your email address and initial password to log in to the labeling platform. After login, change the password. After logging in to the labeling platform, you can view the assigned labeling task and click the task name to go to the labeling page. The labeling method varies depending on the dataset type. For details, see the following:

- **Image Classification**
- **Object Detection**
- **Text Classification**
- **Named Entity Recognition**
- **Text Triplet**

On the labeling platform, each member can view the images that are not labeled, to be corrected, rejected, to be reviewed, approved, and accepted. Pay attention to the images rejected by the administrator and the images to be corrected.

If the Reviewer role is assigned for a team labeling task, the labeling result needs to be reviewed. After the labeling result is reviewed, it is submitted to the administrator for acceptance.

**Figure 2-86 Labeling platform**

**Task Acceptance (Administrator)**

- **Initiating acceptance**

  After team members complete data labeling, the dataset creator can initiate acceptance to check labeling results. The acceptance can be initiated only when a labeling member has labeled data. Otherwise, the acceptance initiation button is unavailable.

  a. On the **Labeling Progress** tab page, click **Initiate Acceptance** to accept tasks.

  b. In the displayed dialog box, set **Sample Policy** to **By percentage** or **By quantity**. Click **OK** to start the acceptance.

  - **By percentage**: Sampling is performed based on a percentage for acceptance.
  - **By quantity**: Sampling is performed based on quantity for acceptance.

**Figure 2-87 Initiating acceptance**

**Initiate Acceptance**

- **Samples for Acceptance**
- **Sampling Policy**
  - **By percentage**
  - **By quantity**

  ![Initiation Acceptance Button](image)
c. After the acceptance is initiated, an acceptance report is displayed on the console in real time. In the **Acceptance Result** area on the right, select **Pass** or **Reject**.

If you select **Pass**, set **Rating** to A, B, C, or D. Option A indicates the highest score. See **Figure 2-89**. If you select **Reject**, enter your rejection reasons in the text box. See **Figure 2-90**.

**Figure 2-88** Viewing a real-time acceptance report

![Figure 2-88](image)

**Figure 2-89** Pass

**Acceptance Result**

**Rating:**

- [ ] A
- [ ] B
- [ ] C
- [ ] D

**Confirm**  **Reject**  **Skip**

**Figure 2-90** Reject

**Acceptance Result**

not good

8/256

**Pass**  **Confirm**  **Skip**

- **Continuing acceptance**

You can continue accepting tasks whose acceptance is not completed. For tasks for which an acceptance process is not initiated, the **Continue Acceptance** button is unavailable.
On the **Labeling Progress** tab page, click **Continue Acceptance** to continue accepting tasks. The **Real-Time Acceptance Report** page is displayed. You can continue to accept the images that are not accepted.

- **Finishing acceptance**

  In the acceptance completion window, you can view dataset acceptance details, such as the number of sample files, set the following parameters, and perform acceptance. The labeling information is synchronized to the **Labeled** tab page of the dataset only after the acceptance is complete.

  Once the labeled data is accepted, team members cannot modify the labeling information. Only the dataset creator can modify the labeling information.

**Table 2-24 Parameters for finishing acceptance**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Modifying Labeled Data |  ● **Not overwrite**: For the same data, do not overwrite the existing data with the labeling result of the current team.  
  ● **Overlays**: For the same data, overwrite the existing data with the labeling result of the current team. Overwritten data cannot be recovered. Exercise caution when performing this operation. |
| Acceptance Scope    |  ● All: all data that has been labeled by the current team, including **Accepted**, **Pending Acceptance**, and **Rejected** data. It refers to all sample files in the dataset.  
  ● All rejects: rejects all data that has been labeled by the current team. That is, all labeled data is rejected to the labeling personnel.  
  ● Accepted and pending acceptance: accepts the data that passes the acceptance or is in the Pending Acceptance state in the sample files and rejects the data that fails the acceptance to the labeling personnel.  
  ● Accepted: accepts the data that has passed the acceptance in the sample files and rejects the data that is in the Pending Acceptance state or fails the acceptance to the labeling personnel. |
Viewing an Acceptance Report

You can view the acceptance report of an ongoing or finished labeling task. On the Labeling Progress tab page, click Acceptance Report. In the displayed Acceptance Report dialog box, view report details.

Deleting a Labeling Task

On the Labeling Progress tab page, click Delete in the row where a labeling task to be deleted. After a task is deleted, the labeling details that are not accepted will be lost. Exercise caution when performing this operation. However, the
original data in the dataset and the labeled data that has been accepted are still stored in the corresponding OBS bucket.
3 DevEnviron (Notebook)

3.1 Introduction to Notebook

ModelArts integrates the open source Jupyter Notebook and JupyterLab to provide you with online interactive development and debugging environments. You can use the Notebook on the ModelArts management console to compile and debug code and train models based on the code, without concerning installation and configurations.

- Jupyter Notebook is an interactive notebook. For details about how to perform operations on Jupyter Notebook, see Jupyter Notebook Documentation.
- JupyterLab is an interactive development environment. It is a next-generation product of Jupyter Notebook. JupyterLab enables you to compile notebooks, operate terminals, edit MarkDown text, open interaction modes, and view CSV files and images. For details about how to perform operations on JupyterLab, see JupyterLab Documentation.

Supported AI Engines

All supported AI engines can be used in the same notebook instance. Different engines can be switched quickly and conveniently, and run in independent development environments. Conda-python2 is a basic Python 2.7 environment without the AI engine. Conda-python3 is a basic Python 3.6 environment without the AI engine.

☐ NOTE

- ModelArts notebook instances support multiple engines. That is, a notebook instance can use all supported engines. Different engines can be switched quickly and conveniently.
- When creating a notebook instance, select a work environment that contains multiple engines, including Python2 and Python3. Multi-Engine 1.0 (Python3) is recommended. Select an applicable work environment. For details, see Table 3-1.
Table 3-1 AI engines

<table>
<thead>
<tr>
<th>Work Environment</th>
<th>Built-in AI Engine and Version</th>
<th>Matching Chip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-Engine 1.0 (Python 3, Recommended)</td>
<td>MXNet-1.2.1</td>
<td>CPU/GPU</td>
</tr>
<tr>
<td></td>
<td>PySpark-2.3.2</td>
<td>CPU</td>
</tr>
<tr>
<td></td>
<td>PyTorch-1.0.0</td>
<td>GPU</td>
</tr>
<tr>
<td></td>
<td>TensorFlow-1.13.1</td>
<td>CPU/GPU</td>
</tr>
<tr>
<td></td>
<td>TensorFlow-1.8</td>
<td>CPU/GPU</td>
</tr>
<tr>
<td></td>
<td>XGBoost-Sklearn</td>
<td>CPU</td>
</tr>
<tr>
<td>Multi-Engine 1.0 (Python 2)</td>
<td>Caffe-1.0.0</td>
<td>CPU/GPU</td>
</tr>
<tr>
<td></td>
<td>MXNet-1.2.1</td>
<td>CPU/GPU</td>
</tr>
<tr>
<td></td>
<td>PySpark-2.3.2</td>
<td>CPU</td>
</tr>
<tr>
<td></td>
<td>PyTorch1.0.0</td>
<td>GPU</td>
</tr>
<tr>
<td></td>
<td>TensorFlow-1.13.1</td>
<td>CPU/GPU</td>
</tr>
<tr>
<td></td>
<td>TensorFlow-1.8</td>
<td>CPU/GPU</td>
</tr>
<tr>
<td></td>
<td>XGBoost-Sklearn</td>
<td>CPU</td>
</tr>
<tr>
<td>Multi-Engine 2.0 (Python 3)</td>
<td>PyTorch-1.4.0</td>
<td>GPU</td>
</tr>
<tr>
<td></td>
<td>R-3.6.1</td>
<td>CPU/GPU</td>
</tr>
<tr>
<td></td>
<td>TensorFlow-2.1.0</td>
<td>CPU/GPU</td>
</tr>
</tbody>
</table>

Constraints

- For security purposes, the root permission is not granted to the notebook instances integrated in ModelArts. You can use the non-privileged user jovyan or ma-user (using Multi-Engine) to perform operations. Therefore, you cannot use apt-get to install the OS software.
- Notebook instances support only standalone training under the current AI engine framework. If you need to use distributed training, you are advised to use ModelArts training jobs and specify multiple nodes in the resource pool.
- ModelArts DevEnviron does not support apt-get. Instead, you can use Custom Image Overview.
- Notebook instances do not support GUI-related libraries, such as PyQt.
- Notebook instances created using Ascend specifications cannot be attached to EVS disks.
- Notebook instances cannot be connected to DWS and database services.
- Notebook instances cannot directly read files in OBS. You need to download the files to the local host. To access data in OBS, you are advised to use MoXing or SDK for interaction.
- DevEnviron does not support TensorBoard. You are advised to use the visualization job function under Training Jobs.

- After a notebook instance is created, you cannot modify its specifications. For example, you cannot change the CPU specifications to GPU specifications or change the work environment. Therefore, you are advised to select the specifications required by the service when creating a notebook instance, or save your code and data to OBS in a timely manner during development so that you can quickly upload the code and data to a new notebook instance.

- If the code output is still displayed after you close the page and open it again, use Terminal.

- After upgrading AI frameworks such as PyTorch, check whether they are compatible with the current CUDA version. The CUDA versions preset in Multi-Engine 2.0 are different from those preset in Multi-Engine 1.0. For details about how to switch CUDA versions, see Switching the CUDA Version on the Terminal Page of a GPU-based Notebook Instance.

### 3.2 Managing Notebook Instances

#### 3.2.1 Creating a Notebook Instance

Before developing a model, create a notebook instance, open it, and perform encoding.

**Background**

- You will be charged for creating and using the notebook instances. The fees vary according to your selected resources. For details about pricing, see Product Pricing Details.

- You will be charged as long as your notebook instance is in the Running status. We recommend you to stop the notebook instance when you no longer need it to avoid unnecessary fees. Alternatively, you can enable the auto stop function when creating a notebook instance so that the notebook instance can automatically stop at the specified time without incurring unnecessary fees.

- Only notebook instances in the Running state can be started.

- A maximum of 10 notebook instances can be created for an account. By default, the notebook instances created by the current user are displayed. If Display Only My Instances is disabled, all notebook instances created by the current account and its IAM users are displayed.

- ModelArts supports multiple AI engines. When creating a notebook instance, select the Python2 or Python3 environment. After the notebook instance is created, access it and create environments with desired AI engines on the Jupyter page.

- If OBS storage is used, ensure that the OBS directory you use and ModelArts are in the same region.

- Before creating a notebook instance, learn about the AI engines supported by ModelArts and their versions.
Creating a Notebook Instance

1. Log in to the ModelArts management console. In the left navigation pane, choose DevEnviron > Notebooks to switch to the Notebooks page.

2. Click Create. On the displayed page, set the required parameters.

   a. Enter the basic information about the notebook instance, including the name, description, and whether to automatically stop the notebook instance. For details about the parameters, see Table 3-2.

   ![](image)

   **Figure 3-1** Basic information about a notebook instance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Billing Mode</td>
<td>Select <strong>Pay-per-use</strong>. Currently, only pay-per-use billing is supported.</td>
</tr>
<tr>
<td>Name</td>
<td>Name of a notebook instance, which contains a maximum of 64 characters. Only digits, uppercase and lowercase letters, underscores (_), and hyphens (-) are allowed. This parameter is mandatory.</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description of a notebook instance.</td>
</tr>
<tr>
<td>Auto Stop</td>
<td>This function is enabled by default. The default value is <strong>1 hour later</strong>, indicating that the notebook instance automatically stops after running for 1 hour and its billing stops. The options are <strong>1 hour later</strong>, <strong>2 hours later</strong>, <strong>4 hours later</strong>, <strong>6 hours later</strong>, and <strong>Custom</strong>. If you select <strong>Custom</strong>, you can specify any integer from 1 to 24 hours.</td>
</tr>
</tbody>
</table>

   b. Set notebook parameters, such as the work environment and instance flavor. For details, see Table 3-3.
Figure 3-2 Notebook instance parameters

Table 3-3 Notebook instance parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Environment</td>
<td>Only public images, that is, the AI frameworks built in ModelArts are supported. All supported AI engines can be used in the same notebook instance. Different engines can be switched quickly and conveniently, and run in independent development environments. After the notebook instance is created, go to the Jupyter page to create the development environment with the desired AI engine. The AI engine varies depending on the operating environment. For details, see Supported AI Engines. <strong>Multi-Engine 1.0 (Python 3, Recommended)</strong> is recommended. If you need to use TensorFlow 2.X, PyTorch 1.4.0, or an AI framework with the R language, you are advised to use <strong>Multi-Engine 2.0 (Python3)</strong>. If you select this work environment, you are advised to select a paid flavor because the free flavor is unavailable. <strong>NOTE</strong> ModelArts also supports the Keras engine. You can select <strong>Multi-Engine 1.0 (Python 3, Recommended)</strong> or <strong>Multi-Engine 1.0 (Python 2)</strong>. For details, see Does ModelArts Support the Keras Engine?</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Resource Pool</td>
<td>Select <strong>Public resource pools</strong> or <strong>Dedicated resource pools</strong>. For details about the dedicated resource pools and how to buy them, see Resource Pools. You do not need to purchase the public resource pool. The public resource pool is available immediately after being provisioned and is charged based on the running duration of your notebook instances. Dedicated resource pools are queue-free. When a large number of users use the public resource pool, they may queue for resources. You are advised to purchase a dedicated resource pool to improve development efficiency.</td>
</tr>
<tr>
<td>Type</td>
<td>Available resource types are CPU and GPU. The GPU type delivers better performance, but is more costly than the CPU type.</td>
</tr>
<tr>
<td>Instance Flavor</td>
<td>A flavor needs to be selected only when you select Public resource pools. The instance flavors vary with the Type setting.</td>
</tr>
<tr>
<td></td>
<td>• If you select CPU for Type, available options include 2 vCPUs</td>
</tr>
<tr>
<td></td>
<td>• If you select GPU for Type, 8 vCPUs</td>
</tr>
<tr>
<td></td>
<td>• If you select GPU for Type, GPU: 1 x P100 CPU: 8 vCPUs</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Storage** | The **EVS** and **OBS** options are available.  
- Selecting **EVS**  
  Set **Disk Space** based on the actual usage. The default value of **Disk Space** is 5 GB. ModelArts provides 5 GB disk space for you to use for free. If the disk space exceeds 5 GB, the additional space is billed by GB according to pricing of **ultra-high I/O** disks. The value of **Disk Space** ranges from 5 to 4096.  
If you select this storage mode, all read and write operations on files on the notebook instances take effect on the data stored in your notebook instances. There is no data loss after you restart notebook instances.  
- Selecting **OBS**  
  Click **Select** next to the **Storage Path** text box to set the OBS path for storing notebook instance data. If you want to use existing files or data, upload the files or data to the corresponding OBS path in advance. **Storage Path** must be set to a specific directory in an OBS bucket rather than the root directory of the OBS bucket.  
If you select this storage mode, all read and write operations on files on the notebook instances take effect on the data stored in your selected OBS path. To synchronize data in a file stored in OBS to a notebook instance, select the file and click **Sync OBS**. For details, see [Synchronizing Files with OBS](#). There is no data loss after you restart this notebook instance. |
| **Git Repository** | This parameter is available only when **EVS** is selected for **Storage**.  
After this function is enabled, you can create a notebook instance with a Git repository. The system automatically synchronizes the code repository from GitHub. For details, see [Creating a Notebook Instance with a Git Repository](#). |

3. Click **Next**.
4. After confirming the parameter configurations, click **Submit**.
   
Switch to the notebook instance list. **Status** of the notebook instance being created is **Starting**. If **Status** of the notebook instance changes to **Running**, the notebook instance has been created.

### 3.2.2 Creating a Notebook Instance with a Git Repository

ModelArts Notebook allows you to download the public and private repositories of GitHub and perform graphical operations based on the Git plug-in of JupyterLab.
Prerequisites

- Currently, the Git repository supports only notebook instances of the EVS type. When creating a notebook instance, you can enable or disable the Git repository only when EVS is selected for Storage. The Git repository is disabled by default.

- Figure 3-3 shows the configuration details after the Git repository is enabled. Git repositories are classified into private repositories and public repositories.

![Figure 3-3 Enabling a Git repository](image)

- The configured code library is downloaded to the corresponding path (/home/ma-user/work) of the notebook instance. You can use the Git plug-in of JupyterLab to start Git use. For details, see Using the Git Plug-in.

- When a notebook instance is stopped and restarted, if a directory with the same project name exists in the /home/ma-user/work directory, the notebook instance will not be downloaded again. In addition, the git_exception.log file is generated in the work directory, indicating that a file with the same name already exists. If the directory does not exist, the notebook instance will be downloaded again after the system is restarted.

Downloading a GitHub Public Repository

After setting Repository Type to Public repository, enter the organization name on GitHub, for example, jupyter, and press Enter or click the search button on the right to view the code library and branch content. If the system responds slowly after you click the search button, wait for a moment.

If the organization has multiple code libraries or branches, you can select the name of the repository to be downloaded from the code library list and select the name of the branch to be downloaded from the branch list. After the configuration is complete, click Next to create a notebook instance.
Figure 3-4 Selecting a code repository and branch

Download a GitHub Private Repository

Currently, a GitHub private repository can be downloaded in OAuth authorization mode.

1. Set **Repository Type** to **Private repository** and click **Authorize with OAuth**.

2. Perform the operations as follows:
   - If you have logged in to GitHub, the authorization page is displayed.
   - If you have not logged in to GitHub, the login page is displayed. Enter the correct username or email address and password and click **Sign in**. After the login is successful, the authentication information is displayed. Click **Authorize ModelArts-Lab** to enable authorization.
When a message is displayed indicating that authorization is created, the GitHub account has been authorized. Manually close the authorization page and go to the Create Notebook page.

3. Click OK.

**NOTE**

If you click Cancel in the dialog box, the OAuth authorization fails and the code of the private repository cannot be downloaded.
4. After the authorization is completed, select the repository and branch under the GitHub account from the drop-down list. Set **Username/Organization**, **Repository Name**, and **Branch** in sequence. The setting of a parameter affects the value of another parameter. If the drop-down list does not contain anything, wait 1 to 2 minutes and set the parameter again.

**Figure 3-8** Selecting the username, repository name, and branch

5. Click **Next** to create a notebook instance.

**Viewing a Notebook with a Git Repository**

After a notebook instance with a Git repository is created, you can view the repository in different locations, regardless of whether the repository is a public repository or private repository.

- View the repository address in the notebook list.
  
  In the notebook instance list, click the triangle on the left of the notebook instance name to expand the instance details and view the repository address of the notebook instance.

**Figure 3-9** Viewing the repository address
Open the notebook instance to view the code library. After the notebook instance is started up and opened, you can view the code library. If the code library is downloaded successfully, open the running notebook instance and view the code library on the Jupyter page.

**Figure 3-10** Viewing the code library

If the page shown in **Figure 3-11** is displayed when a project is opened, the `git.log` file is contained. Open the `git.log` file. If the file content is the same as that shown in **Figure 3-12**, the repository is being downloaded. The download time varies depending on the repository size and network conditions. If the file content is different from that shown in **Figure 3-12**, rectify the fault as prompted or submit a service ticket.

**Figure 3-11** Repository being downloaded

**Figure 3-12** Content of the `git.log` file

### 3.2.3 Opening a Notebook Instance

You can open a created notebook instance (that is, an instance in the **Running** state) and start coding in the development environment.

#### Instance Opening

- **Go to the Jupyter Notebook page.**
  
  In the notebook instance list, locate the row where the target notebook instance resides and click **Open** in the **Operation** column or click the notebook instance name.

- **Go to the JupyterLab page.**
  
  In the notebook instance list, select the notebook instance to be opened and click **Open JupyterLab** in the **Operation** column.
Code Development

ModelArts provides two environments for code development: Jupyter Notebook and JupyterLab.

- **Jupyter Notebook**: a web-based application for interactive computing. It can be applied to full-process computing: development, documentation, running code, and presenting results.
- **JupyterLab**: an interactive development environment. It is a next-generation product of Jupyter Notebook. JupyterLab enables you to compile notebooks, operate terminals, edit MarkDown text, open interaction modes, and view CSV files and images.

### 3.2.4 Starting or Stopping a Notebook Instance

You can stop unwanted notebook instances to avoid unnecessary fees. You can also start a notebook instance that is in the **Stopped** state to use it again.

Log in to the ModelArts management console. In the left navigation pane, choose DevEnviron > Notebooks to switch to the Notebooks page. Perform the following operations to stop or start a notebook instance:

- To stop a notebook instance, locate the row where the notebook instance resides and click **Stop** in the **Operation** column. Only notebook instances in the **Running** state can be stopped.
- To start a notebook instance, locate the row where the notebook instance resides and click **Start** in the **Operation** column. Only notebook instances in the **Stopped** state can be started.

### 3.2.5 Deleting a Notebook Instance

You can delete notebook instances that are no longer used to release resources.

1. Log in to the ModelArts management console. In the left navigation pane, choose DevEnviron > Notebooks to switch to the Notebooks page.
2. In the notebook instance list, locate the row where the target notebook instance resides and click **Delete** in the **Operation** column. In the dialog box that is displayed, click **OK**.

**NOTE**

Deleted notebook instances cannot be recovered. Therefore, exercise caution when performing this operation. However, the files created in notebook instances are still stored in OBS specified during creation of the notebook instances.

If the notebook instance uses EVS storage, data will be deleted together with the notebook instance. Exercise caution when deleting the notebook instance.

### 3.3 Using Jupyter Notebook
3.3.1 Introduction to Jupyter Notebook

Jupyter Notebook is a web-based application for interactive computing. It can be applied to full-process computing: development, documentation, running code, and presenting results.

ModelArts integrates the open source Jupyter Notebook. After creating a notebook instance, you can open the instance for development without the need for installation and configuration.

Notebook Kernel

- A notebook kernel is an independent code execution environment. The ModelArts Notebook function provides multiple kernel types, such as TensorFlow 1.13.1 and PyTorch 1.0. A code execution environment contains the pre-installed and commissioned AI engines and dependencies.
- When a kernel is selected to open a notebook instance, an IPython process is started at the backend of the notebook instance as the running environment to execute the code and command input on the page.
- Each kernel type contains an independent Conda running environment to ensure that the AI engines are independent of each other. For example, if the Keras library is updated in a kernel of the TensorFlow type, the kernel of the PyTorch type is not affected.

Differences Between Notebook Kernels and Common Interactive Python Interpreters

A notebook kernel is an IPython running environment, which can be considered as an enhanced Python shell. Compared with a Python interpreter, a notebook kernel can execute shell scripts and integrate more visualized tools and magic commands. For details, see IPython Documentation.

3.3.2 Common Operations on Jupyter Notebook

This section describes common operations on Jupyter Notebook.

Opening Jupyter Notebook

In the notebook instance list, locate the row where the target notebook instance resides and click Open in the Operation column to switch to the Jupyter Notebook page.

Three tab pages are available on the Jupyter page: Files, Running, and ModelArts Examples

Figure 3-14 Jupyter Notebook

Selecting Different AI Engines to Create Files

Open a notebook instance and go to the Jupyter Notebook page. On the Files tab page, click New in the upper right corner, select the required AI engine, and create a file for encoding.

**Figure 3-15 Selecting different AI engines**

![Selecting different AI engines](image)

Uploading a File

Open a notebook instance and go to the Jupyter Notebook page. On the Files tab page, click Upload in the upper right corner to select a file from the local PC and upload it.

The size of the file to be uploaded using this function is limited. If the file size exceeds the limit, you are advised to use other methods to upload the file. For details, see **Uploading Large Files to a Notebook Instance**.

**Figure 3-16 Uploading a file**

![Uploading a file](image)

Compiling a File

After a file is created, click the file name to go to the file compilation page.
Table 3-4 Introduction to the file compilation page

<table>
<thead>
<tr>
<th>No.</th>
<th>Area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>File name</td>
<td>You can enter a user-defined file name in this area. After the file name is changed and saved, the new file name is updated to the file list accordingly.</td>
</tr>
<tr>
<td>2</td>
<td>Menu bar</td>
<td>The menu bar provides rich functions such as File, Edit, View, Insert, Cell, Kernel, and Help. For details, see Jupyter Notebook Documentation. The following toolbar provides common functions for compiling common Python running files.</td>
</tr>
</tbody>
</table>
| 3   | Toolbar             | The toolbar lists the common shortcut operations. From left to right, the shortcut operations are as follows: saving a file, adding a new cell, cutting a selected cell, copying a selected cell, pasting a selected cell, moving a selected cell upwards, moving a selected cell downwards, running a selected cell, terminating the kernel, restarting the kernel, and restarting the kernel and running all cells again. The Code drop-down list contains the following options:  
  ● Code: Write Python code.  
  ● MarkDown: Write MarkDown code, which is usually used for comments.  
  ● Raw NBConvert: conversion tool.  
  ● Heading: Quickly add a MarkDown title. |
| 4   | AI engine and Python version | Displays the AI engine and Python version corresponding to the current file. For details about all AI engines and Python versions supported by ModelArts, see Supported AI Engines. |
### Deleting a File or Folder

To delete a file or folder from Jupyter Notebook, select the file or folder in the \textbf{Files} list and click the \textbf{Delete} button.

\begin{quote}
\textbf{NOTE}

After the file or folder is deleted, click the \textbf{Refresh} button in the upper right corner to refresh the Jupyter page and clear the cache.
\end{quote}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{jupyter_page.png}
\caption{Jupyter page}
\end{figure}

### 3.3.3 Configuring the Jupyter Notebook Environment

#### 3.3.3.1 Using the Notebook Terminal Function

For developers who are used to coding, the terminal function is very convenient and practical. This section describes how to enable the terminal function in a notebook instance and switch the engine environment in the terminal.

\begin{quote}
\textbf{Enabling the Notebook Terminal Function}

1. In the notebook instance list, click \textbf{Open} in the \textbf{Operation} column of the target notebook instance to go to the \textbf{Jupyter Notebook} page.
\end{quote}
2. On the Files tab page of the Jupyter page, click New and select Terminal. The Terminal page is displayed.

**Figure 3-19** Going to the Terminal page

Switching Engine Environments on the Terminal

You can switch to another AI engine environment in the terminal environment of Jupyter.

1. Create and open a notebook instance or open an existing notebook instance in the notebook instance list.
2. On the Files tab page of the Jupyter page, click New and select Terminal. The Terminal page is displayed.
3. You can switch environments by following README in the current directory. For example, run the `source /home/ma-user/anaconda3/bin/activate TensorFlow-1.8` command to switch to the TensorFlow-1.8 environment and develop a notebook instance. To exit, run the `source deactivate` command.

**Figure 3-20** Running the command

3.3.3.2 Switching the CUDA Version on the Terminal Page of a GPU-based Notebook Instance

For a GPU-based notebook instance, you can switch different versions of CUDA on the Terminal page of Jupyter.

CPU-based notebook instances do not use CUDA. Therefore, the following operations apply only to GPU-based notebook instances.

1. Create and open a notebook instance or open an existing notebook instance in the notebook instance list.
2. On the Files tab page of the Jupyter page, click New and select Terminal. The Terminal page is displayed.
3. Run the following command to go to `/usr/local`:
   ```bash
cd /usr/local
   ```

4. For example, to switch to CUDA 10, run the following command:
   ```bash
   sudo ln -snf /usr/local/cuda-10.0 /usr/local/cuda
   ```

**Figure 3-21** Example of switching the CUDA version

---

### 3.3.3.3 Installing External Libraries and Kernels in Notebook Instances

Multiple environments have been installed in ModelArts notebook instances. These environments contain Jupyter and Python packages, including TensorFlow, MXNet, Caffe, PyTorch, and Spark. You can use `pip install` to install external libraries from a Jupyter notebook or terminal to facilitate use.

#### Installing an External Library from a Jupyter Notebook

Assume that you want to install Shapely from a notebook instance. Follow the following instructions:

1. In the left navigation pane of the ModelArts management console, choose **DevEnviron > Notebooks**. Open a notebook instance in the displayed notebook instance list.
2. In the Jupyter Notebook page that is displayed, click **New** and select the required AI engine from the drop-down list.
3. In the displayed window, type the following command in the code input bar to install Shapely:
   ```bash
   pip install shapely
   ```

#### Installing an External Library from a Terminal

Assume that you want to install Shapely from the terminal of a notebook instance by using `pip`. Follow the following instructions:

1. In the left navigation pane of the ModelArts management console, choose **DevEnviron > Notebooks**. Open a notebook instance in the displayed notebook instance list.
2. In the displayed Jupyter dashboard, click **New** and choose **Terminal** from the shortcut menu.
3. For a notebook instance that does not use the AI engine of the Multi-Engine type, enter the following command in the code input bar to install Shapely:

```
/opt/conda/envs/python27_tf/bin/pip install Shapely
```

4. The Multi-Engine notebook instance can use multiple engines. By referring to the README file in the /home/ma-user/ path, switch to the installation package of the corresponding engine environment and install Shapely. For example, you can install Shapely from TensorFlow-1.13.1 with the following code:

```
source /home/ma-user/anaconda3/bin/activate TensorFlow-1.13.1
pip install shapely
```

Table 3-5 lists the Python paths of the TensorFlow, MXNet, PyTorch, Caffe, Scikit-learn & XGBoost, and Spark engines in the terminal. The pip system is also installed in the same directory as the related engine. For details about the engines used by Multi-Engine notebook instances, refer to the README file.

<table>
<thead>
<tr>
<th>AI Engine</th>
<th>Version</th>
<th>Python Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>TF-1.8.0-python2.7</td>
<td>/opt/conda/envs/python27_tf/bin/python</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>TF-1.8.0-python3.6</td>
<td>/opt/conda/envs/python36_tf/bin/python</td>
</tr>
<tr>
<td>MXNet</td>
<td>MXNet-1.2.1-python2.7</td>
<td>/opt/conda/envs/python27_mxnet/bin/python</td>
</tr>
<tr>
<td>MXNet</td>
<td>MXNet-1.2.1-python3.6</td>
<td>/opt/conda/envs/python36_mxnet/bin/python</td>
</tr>
<tr>
<td>PyTorch</td>
<td>PyTorch-1.0.0-python2.7</td>
<td>/opt/conda/envs/python27_pytorch/bin/python</td>
</tr>
<tr>
<td>PyTorch</td>
<td>PyTorch-1.0.0-python3.6</td>
<td>/opt/conda/envs/python36_pytorch/bin/python</td>
</tr>
<tr>
<td>Caffe</td>
<td>Caffe-1.0.0-python2.7</td>
<td>/opt/conda/envs/python27_caffe/bin/python</td>
</tr>
<tr>
<td>Scikit-learn &amp; XGBoost</td>
<td>ML-1.0.0-python2.7</td>
<td>/opt/notebook/anaconda2/bin/python</td>
</tr>
<tr>
<td>Spark</td>
<td>Spark-2.2.0-python2.7</td>
<td></td>
</tr>
<tr>
<td>Scikit-learn &amp; XGBoost</td>
<td>ML-1.0.0-python3.6</td>
<td>/opt/notebook/anaconda3/bin/python</td>
</tr>
<tr>
<td>Spark</td>
<td>Spark-2.2.0-python3.6</td>
<td></td>
</tr>
</tbody>
</table>
When you create a ModelArts training job, a new independent running environment is started, which is not associated with the packages installed in the Notebook environment. Therefore, add `os.system('pip install xxx')` to the startup code before importing the installation package.

For example, if you need to use the Shapely dependency in the training job, add the following code to the startup code:

```python
os.system('pip install Shapely')
import Shapely
```

### 3.3.4 Uploading and Downloading Data

#### 3.3.4.1 Uploading Large Files to a Notebook Instance

On the Notebook page, click **Upload** to upload a file. For details, see Uploading a File in Common Operations on Jupyter Notebook. If a message is displayed indicating that the size of the files to be uploaded exceeds the upper limit when uploading files to notebook instances or JupyterLab, you can upload the files to OBS and then download them to notebook instances.

**Step 1: Uploading Files to OBS**

You are advised to use an OBS tool, such as OBS Browser+ or obsutil, to upload large files because OBS Console has restrictions on the file size and quantity. OBS Browser+ is a graphical tool that provides complete functions for managing your buckets and objects in OBS. You are advised to use this tool to create buckets or upload objects. obsutil is a command line tool for accessing and managing OBS resources. If you are familiar with command line interface (CLI), obsutil is recommended as an ideal tool for batch processing and automated tasks. For details about how to upload files to OBS, see Object Storage Service Tools Guide.

**Step 2: Downloading Files from OBS to Notebook Instances**

A notebook instance can be mounted to OBS or EVS as the storage location. The operation method varies depending on the instance types.

- **Downloading files to notebook instances with EVS attached**
  - Use the following MoXing API to synchronize files from OBS to notebook instances.

```python
file_str = mox.file.read('obs://bucket_name/obs_file.txt')
```

You can also open the file object and read data from it. Both methods are equivalent.

```python
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
    file_str = f.read()
```

- Use the OBS API in the ModelArts SDK to download data **from OBS** to notebook instances.
If the size of a single file exceeds 5 GB, the file cannot be uploaded in this mode. You are advised to use the MoXing API to upload large files.

Sample code:
```python
from modelarts.session import Session
session = Session()
session.download_data(bucket_path="/bucket-name/dir1/sdk.txt", path="/home/user/sdk/obs.txt")
```

- Downloading files to notebook instances using OBS for data storage
  Upload files to the OBS path specified during notebook instance creation and synchronize the files from OBS to the notebook instances using Sync OBS.

### 3.3.4.2 Downloading Large Files in Notebook Instances to a Local PC

Only files within 100 MB in notebook instances can be downloaded to a local PC. You can perform operations in different scenarios based on the storage location selected when creating a notebook instance.

#### Notebook Instances with EVS Attached

For notebook instances with EVS attached, you can perform the following operations to download large files to the local PC:

1. In the notebook instance, create an ipynb file. Use MoXing to upload the large files from notebook instances to OBS. The sample code is as follows:
   ```python
   import moxing as mox
   mox.file.copy('/home/ma-user/work/obs_file.txt', 'obs://bucket_name/obs_file.txt')
   
   In the preceding code, `/home/ma-user/work/obs_file.txt` indicates a file storage path in a notebook instance, and `obs://bucket_name/obs_file.txt` indicates a file storage path on OBS.

2. Use OBS or the ModelArts SDKs to download the files from OBS to the local PC.
   - Method 1: Downloading the files using OBS
     Use OBS to download the files. You can download the `obs_file.txt` file in the example to the local PC. If you have a large amount of data, you are advised to use OBS Browser+ to download data or folders.
   - Method 2: Downloading the files using the ModelArts SDKs
     i. Download and install the ModelArts SDKs on your local PC. For details, see [SDK Download](#).
     ii. Authenticate sessions of the ModelArts SDKs. For details, see [Session Authentication](#).
     iii. Download the files from OBS to the local PC. The sample code is as follows:
        ```python
        from modelarts.session import Session
        session = Session(access_key='***', secret_key='***', project_id='***', region_name='***')
        session.download_data(bucket_path="/bucket_name/obs_file.txt", path="/home/user/obs_file.txt")
        ```

#### Notebook Instances Using OBS Storage

For notebook instances that use OBS storage, you can use OBS or the ModelArts SDK to download files from OBS to a local PC.
Method 1: Downloading the files using OBS

Use OBS to **download the files** to the local PC. If you have a large amount of data, you are advised to use OBS Browser+ to download data or folders.

Method 2: Downloading the files using the ModelArts SDKs

a. Download and install the ModelArts SDKs on your local PC. For details, see **SDK Download**.

b. Authenticate sessions of the ModelArts SDKs. For details, see **Session Authentication**.

c. Download the files from OBS to the local PC. The sample code is as follows:

```python
from modelarts.session import Session
session = Session(access_key='***', secret_key='***', project_id='***', region_name='***')
session.download_data(bucket_path='/bucket_name/obs_file.txt', path='/home/user/obs_file.txt')
```

### 3.3.5 Using ModelArts SDKs

In notebook instances, you can use ModelArts SDKs to manage OBS, training jobs, models, and real-time services.

For details about how to use ModelArts SDKs, see the [ModelArts SDK Reference](#).

Notebooks carry the authentication (AK/SK) and region information about login users. Therefore, SDK session authentication can be completed without entering parameters.

#### Example Code

- **Creating a training job**

```python
from modelarts.session import Session
from modelarts.estimator import Estimator
session = Session()
estimator = Estimator(modelarts_session=session,  # AI engine name
                     framework_type='PyTorch',
                     framework_version='PyTorch-1.0.0-python3.6',
                     code_dir='/obs-bucket-name/src/',  # AI engine version
                     boot_file='/obs-bucket-name/src/pytorch_sentiment.py',  # Training script directory
                     log_url='/obs-bucket-name/log/',  # Training startup
                     hyperparameters=[
                         {'label': 'classes',
                          'value': '10'},
                         {'label': 'lr',
                          'value': '0.001'}
                     ],
                     output_path='/obs-bucket-name/output/',  # Training output
                     train_instance_type='modelarts.vm.gpu.p100',  # Training environment
                     train_instance_count=1,  # Number of training nodes
                     job_description='pytorch-sentiment with ModelArts SDK')  # Training job description
job_instance = estimator.fit(inputs='/obs-bucket-name/data/train/',
                             wait=False,
                             job_name='my_training_job')
```

- **Querying a model list**

```python
from modelarts.session import Session
from modelarts.model import Model
session = Session()
model_list_resp = Model.get_model_list(session, model_status='published', model_name='digit', order='desc')
```
● Querying service details
from modelarts.session import Session
from modelarts.model import Predictor
session = Session()
predictor_instance = Predictor(session, service_id="input your service_id")
predictor_info_resp = predictor_instance.get_service_info()

3.3.6 Synchronizing Files with OBS

If you specify **Storage Path** during notebook instance creation, your compiled code will be automatically stored in your specified OBS bucket. If code invocation among different .ipynb files is required, you can use the Sync OBS function.

The Sync OBS function is used to synchronize the objects selected in the list of notebook instance files from the OBS bucket to the current container directory ~/work.

**Precautions**

- The maximum size of files to be synchronized at a time is 500 MB, and the maximum number of files to be synchronized at a time is 1,024.
- The total size of objects to be synchronized cannot exceed 5 GB. For example, if 2 GB files exist in the ~/work container directory, you can use Sync OBS to synchronize a maximum of 3 GB files.
- The Sync OBS function only takes effect on notebook instances for which **Storage is OBS**. For notebook instances whose **Storage** is not **OBS**, all files are read and written in the ~/work container directory.
- Notebook instances created using Ascend resources do not support the Sync OBS function.

**Procedure**

The Sync OBS function can be used in notebook instances. The following describes how to use the function.

For example, assume that the **Untitled2.ipynb** file needs to call **module** in the **Untitled3.ipynb** file. Select both files and click **Sync OBS**.

**Figure 3-22 Using the Sync OBS function**
3.3.7 Using the Convert to Python File Function

After code compiling is finished, you can save the entered code as a .py file which can be used for starting training jobs.

1. Create and open a notebook instance or open an existing notebook instance in the notebook instance list.
2. On the Files tab page, click New and choose the required AI engine from the drop-down list to access the code development page.
3. After code compiling is complete, click the save button in the upper left corner. Then, click Convert to Python File to convert the current ipynb file into a Python file. This function can be used to directly save your entered code as a .py file to the working directory.

The generated .py file can be used to start ModelArts training jobs.

Figure 3-23 Convert to Python File

4. In the dialog box that is displayed, enter the file name as required, and select or deselect Force overwrite if file already exists. By default, the item is not selected, indicating that the file will not be overwritten when a file with the same name exists in the directory. Click Convert.

Figure 3-24 Setting and saving the configuration

3.4 Using JupyterLab
3.4.1 JupyterLab Overview and Common Operations

JupyterLab is an interactive development environment. It is a next-generation product of Jupyter Notebook. JupyterLab enables you to compile notebooks, operate terminals, edit MarkDown text, open interaction modes, and view CSV files and images.

JupyterLab will be a mainstream development environment for developers. JupyterLab supports more flexible and powerful project operations, but has the same components as Jupyter Notebook.

ModelArts supports Jupyter Notebook and JupyterLab. You can use different tools to develop code in the same notebook instance.

Opening JupyterLab

1. Log in to the ModelArts management console. In the left navigation pane, choose DevEnviron > Notebooks to switch to the Notebooks page.
2. Select a notebook instance in the Running state and click Open in the Operation column to access the notebook instance.
3. On the Jupyter page, click Open JupyterLab in the upper right corner to access the JupyterLab page of the notebook instance.

4. The Launcher page is automatically displayed, as shown in the following figure. You can use all open source functions. For details, see JupyterLab Documentation.

Figure 3-25 Accessing JupyterLab

Figure 3-26 JupyterLab home page
Creating and Opening a Notebook Instance

On the JupyterLab home page, click an applicable AI engine in the Notebook area to create a notebook file with the corresponding framework.

The AI framework supported by each notebook instance varies according to the working environment. The following figure is only an example. Select an AI framework based on the site requirements. For details about all framework versions and Python versions supported by ModelArts, see Supported AI Engines.

**Figure 3-27** Selecting an AI engine and creating a notebook instance

The created notebook file is displayed in the navigation tree on the left.

**Figure 3-28** Creating a notebook file

Creating a Notebook File and Opening the Console

A console is essentially a Python terminal. After a statement is entered, the corresponding output is displayed, which is similar to the native IDE of Python.

On the JupyterLab home page, click an applicable AI engine in the Console area to create a notebook file with the corresponding framework.

The AI framework supported by each notebook instance varies according to the working environment. The following figure is only an example. Select an AI framework based on the site requirements.

**Figure 3-29** Selecting an AI engine and creating a console

After the file is created, the console page is displayed.
Uploading a File

On the JupyterLab page, you can click **Upload File** in the upper left corner and select a local file to upload.

The size of the file to be uploaded using this function is limited. If the file size exceeds the limit, you are advised to use other methods to upload the file. For details, see **Uploading Data to JupyterLab**.

Editing a File

JupyterLab allows you to open multiple notebook instances or files (such as HTML, TXT, and Markdown files) in the same window and display them on different tab pages.
Using JupyterLab, you can orchestrate multiple files. In the file display area on the right, you can drag a file to adjust its position. You can open multiple files at the same time.

**Figure 3-32 Orchestration of multiple files**

![Image of JupyterLab with multiple files](image)

When writing code in a notebook instance, you can create multiple views of a file to synchronously edit the file and view the execution result in real time.

To open multiple views, open the file and choose **File > New View for Notebook**.

**Figure 3-33 Multiple views of a file**

![Image of JupyterLab with multiple views](image)
Common Icons and Plug-ins of JupyterLab

**Figure 3-34** Common icons and plug-ins of JupyterLab

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Opens the Launcher page. Then you can quickly create notebook instances, consoles, or other files.</td>
</tr>
<tr>
<td></td>
<td>Creates a folder.</td>
</tr>
<tr>
<td></td>
<td>Uploads a file. For details, see <strong>Uploading a File</strong>.</td>
</tr>
<tr>
<td></td>
<td>Updates a folder.</td>
</tr>
</tbody>
</table>
3.4.2 Uploading and Downloading Data

3.4.2.1 Uploading Data to JupyterLab

On the JupyterLab page, click Upload Files to upload a file. For details, see Uploading a File in JupyterLab Overview and Common Operations. If a message is displayed indicating that the size of the files to be uploaded exceeds the upper limit when uploading files to notebook instances or JupyterLab, you can upload the files to OBS and then download them to notebook instances.

Step 1: Uploading Files to OBS

You are advised to use an OBS tool, such as OBS Browser+ or obsutil, to upload large files because OBS Console has restrictions on the file size and quantity. OBS Browser+ is a graphical tool that provides complete functions for managing your buckets and objects in OBS. You are advised to use this tool to create buckets or

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon]</td>
<td>Git plug-in, which can be used to connect to the GitHub code library associated with the notebook instance. For details, see Using the Git Plug-in.</td>
</tr>
</tbody>
</table>

Table 3-7 Common plug-ins in the plug-in area

<table>
<thead>
<tr>
<th>Plug-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon]</td>
<td>Lists files. You can click here to display the list of all files in the notebook instance.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Lists ModelArts examples. You can click any example in the list to view its code and version mapping.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Displays the terminals and kernels that are running in the current instance.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Git plug-in, which can be used to quickly use the GitHub code library. For details, see Using the Git Plug-in.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Quick start command.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Displays the tab page listing the files that are being opened.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>Document organization.</td>
</tr>
</tbody>
</table>
upload objects. obsutil is a command line tool for accessing and managing OBS resources. If you are familiar with command line interface (CLI), obsutil is recommended as an ideal tool for batch processing and automated tasks. For details about how to upload files to OBS, see Object Storage Service Tools Guide.

Step 2: Downloading Files from OBS to Notebook Instances

A notebook instance can be mounted to OBS or EVS as the storage location. The operation method varies depending on the instance types.

- Downloading files to notebook instances with EVS attached
  - Use the following MoXing API to synchronize files from OBS to notebook instances.

Read an OBS file. For example, if you read the `obs://bucket_name/obs_file.txt` file, the content is returned as strings.

```python
file_str = mox.file.read('obs://bucket_name/obs_file.txt')
```

You can also open the file object and read data from it. Both methods are equivalent.

```python
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
    file_str = f.read()
```

- Use the OBS API in the ModelArts SDK to download data from OBS to notebook instances.

  **NOTE**

  If the size of a single file exceeds 5 GB, the file cannot be uploaded in this mode. You are advised to use the MoXing API to upload large files.

Sample code:

```python
from modelarts.session import Session
session = Session()
session.download_data(bucket_path="/bucket-name/dir1/sdk.txt", path="/home/user/sdk/obs.txt")
```

- Downloading files to notebook instances using OBS for data storage
  Upload files to the OBS path specified during notebook instance creation and synchronize the files from OBS to the notebook instances using Sync OBS.

3.4.2.2 Downloading a File from JupyterLab

Only files within 100 MB in JupyterLab can be downloaded to a local PC. You can perform operations in different scenarios based on the storage location selected when creating a notebook instance.

Notebook Instances with EVS Attached

For notebook instances with EVS attached, you can perform the following operations to download large files to the local PC:

1. In the notebook instance, create an ipynb file. Use MoXing to upload the large files from notebook instances to OBS. The sample code is as follows:

   ```python
   import moxing as mox
   mox.file.copy('/home/ma-user/work/obs_file.txt', 'obs://bucket_name/obs_file.txt')
   ```

   In the preceding code, `/home/ma-user/work/obs_file.txt` indicates a file storage path in a notebook instance, and `obs://bucket_name/obs_file.txt` indicates a file storage path on OBS.
2. Use OBS or the ModelArts SDKs to download the files from OBS to the local PC.
   - Method 1: Downloading the files using OBS
     Use OBS to **download the files**. You can download the **obs_file.txt** file in the example to the local PC. If you have a large amount of data, you are advised to use OBS Browser+ to download data or folders.
   - Method 2: Downloading the files using the ModelArts SDKs
     i. Download and install the ModelArts SDKs on your local PC. For details, see **SDK Download**.
     ii. Authenticate sessions of the ModelArts SDKs. For details, see **Session Authentication**.
     iii. **Download the files** from OBS to the local PC. The sample code is as follows:

     ```python
     from modelarts.session import Session
     session=Session(access_key='***',secret_key='***',project_id='***',region_name='***')
     session.download_data(bucket_path="/bucket_name/obs_file.txt",path="/home/user/obs_file.txt")
     ```

**Notebook Instances Using OBS Storage**

For notebook instances that use OBS storage, you can use OBS or the ModelArts SDK to download files from OBS to a local PC.

- Method 1: Downloading the files using OBS
  Use OBS to **download the files** to the local PC. If you have a large amount of data, you are advised to use OBS Browser+ to download data or folders.

- Method 2: Downloading the files using the ModelArts SDKs
  a. Download and install the ModelArts SDKs on your local PC. For details, see **SDK Download**.
  b. Authenticate sessions of the ModelArts SDKs. For details, see **Session Authentication**.
  c. **Download the files** from OBS to the local PC. The sample code is as follows:

     ```python
     from modelarts.session import Session
     session=Session(access_key='***',secret_key='***',project_id='***',region_name='***')
     session.download_data(bucket_path="/bucket_name/obs_file.txt",path="/home/user/obs_file.txt")
     ```

### 3.4.3 Using Sample Notebook Instances of ModelArts

ModelArts provides sample Jupyter notebooks to help beginners learn how to use ModelArts notebook instances.

**Previewing Sample Notebook Instances of ModelArts**

In JupyterLab, click the second button on the left, that is, **ModelArts Examples**. Double-click an example to open it. The opened example is displayed on the right of JupyterLab. The opened example is read-only and can be viewed.
Operations Allowed on the ModelArts Examples Tab Page

In the Examples list, click an example to open it. The opened example is displayed on the right of JupyterLab. The opened example is read-only and can be viewed. If you want to edit or run the example on the local host, click Create a copy in the upper right corner of the page and enter the path for saving the example. The example and dependency files are copied to the path, and a new tab page is displayed for you to edit and run the example.

3.4.4 Using ModelArts SDKs

In notebook instances, you can use ModelArts SDKs to manage OBS, training jobs, models, and real-time services.

For details about how to use ModelArts SDKs, see the ModelArts SDK Reference.
Notebooks carry the authentication (AK/SK) and region information about login users. Therefore, SDK session authentication can be completed without entering parameters.

### Example Code

#### Creating a training job

```python
from modelarts.session import Session
from modelarts.estimator import Estimator

session = Session()
estimator = Estimator(modelarts_session=session,
    framework_type='PyTorch',  # AI engine name
    framework_version='PyTorch-1.0.0-python3.6',  # AI engine version
    code_dir='/obs-bucket-name/src/',  # Training script directory
    boot_file='/obs-bucket-name/src/pytorch_sentiment.py',  # Training startup script directory
    log_url='/obs-bucket-name/log/',  # Training log directory
    hyperparameters=[
        {'label':'classes', 'value': '10'},
        {'label': 'lr', 'value': '0.001'}
    ],
    output_path='/obs-bucket-name/output/',  # Training output directory
    train_instance_type='modelarts.vm.gpu.p100',  # Training environment specifications
    train_instance_count=1,  # Number of training nodes
    job_description='pytorch-sentiment with ModelArts SDK')

job_instance = estimator.fit(inputs='/obs-bucket-name/data/train/',
    wait=False,
    job_name='my_training_job')
```

#### Querying a model list

```python
from modelarts.session import Session
from modelarts.model import Model

session = Session()
model_list_resp = Model.get_model_list(session, model_status="published", model_name="digit", order="desc")
```

#### Querying service details

```python
from modelarts.session import Session
from modelarts.model import Predictor

session = Session()
predictor_instance = Predictor(session, service_id="input your service_id")
predictor_info_resp = predictor_instance.get_service_info()
```

### 3.4.5 Synchronizing Files with OBS

If you specify **Storage Path** during notebook instance creation, your compiled code will be automatically stored in your specified OBS bucket. If code invocation among different .ipynb files is required, you can use the Sync OBS function.

The Sync OBS function is used to synchronize the objects selected in the list of notebook instance files from the OBS bucket to the current container directory `~/.work`.

### Precautions

- The maximum size of files to be synchronized at a time is 500 MB, and the maximum number of files to be synchronized at a time is 1,024.
- The total size of objects to be synchronized cannot exceed 5 GB. For example, if 2 GB files exist in the `~/.work` container directory, you can use Sync OBS to synchronize a maximum of 3 GB files.
- The Sync OBS function only takes effect on notebook instances for which Storage is OBS. For notebook instances whose Storage is not OBS, all files are read and written in the ~/work container directory.
- Notebook instances created using Ascend resources do not support the Sync OBS function.

**Procedure**

The Sync OBS function can be used in JupyterLab. The following describes how to use the function.

For example, **Example-1.ipynb** needs to invoke module in **Example-2.ipynb**. In **File Browser** on the left, select the two .ipynb files and click **Sync OBS**. In the dialog box that is displayed, click **YES** to complete synchronization. After the synchronization is successful, the code can be called mutually.

**Figure 3-36 Using the Sync OBS function**

3.4.6 Using the Git Plug-in

For notebook instances with the GitHub code library, you can use the Git plug-in on the JupyterLab page to quickly view the content and submit the modified code.

**Prerequisites**

A notebook instance with a Git repository has been created and is in the **Running** state.
Opening the Git Plug-in of JupyterLab

1. In the notebook instance list, select an instance with a Git repository and click the name to open the notebook instance.

2. On the Jupyter page, click Open JupyterLab in the upper right corner. The JupyterLab page is displayed. Figure 3-37 shows the Git plug-in of JupyterLab.

![Figure 3-37 Git plug-in](image)

Viewing Code Library Information

In the list under Name, double-click the folder you want to use and click the Git plug-in icon on the left to access the code library corresponding to the folder.
You can view the information about the current code library, such as the repository name, branch, and historical submission records.
Viewing Content Changes

If you modify a file in the code library, you can view the modified file under Changed on the Changes tab page. You can click Diff this file on the right of the file name to view the changes to the file content.

Committing Content Changes

After confirming that the changes are correct, click Stage this change (equivalent to running the git add command) on the right of the file name. The file enters the
Staged state. Enter the message to be committed in the lower left corner and click Commit (equivalent to running the git commit command).

Figure 3-41 Committing content changes

On the History tab page, you can see that the committing is successful.

Figure 3-42 Checking whether the committing is successful

Clicking the push button is equivalent to running the git push command to push the code to the GitHub repository. After the pushing is successful, the message "Git Push completed successfully" is displayed. If the token used for OAuth authentication has expired, a dialog box is displayed asking you to enter the user token or account information. Enter the information as prompted.
After the preceding operations are completed, on the History tab page of the Git plug-in page of JupyterLab, you can see that origin/HEAD and origin/master point to the latest push. In addition, you can find the corresponding information in the committing records of the GitHub repository.

### 3.4.7 Setting the Initialization Script

When using a notebook instance for development, users often install some software. However, after the notebook instance is stopped and then restarted, the software needs to be reinstalled, which affects the efficiency. The function of setting the boot initialization script is added to JupyterLab of DevEnviron. You can integrate the package installation operations into the script and set the script as the boot initialization script. Then, this script is executed by default each time the notebook instance is started, improving the development efficiency.

#### Prerequisites

The script has been encoded in JupyterLab.

#### Constraints

Package installation operations that must be performed by the root user, for example, `apt-get` package installation, cannot be performed.

#### Setting the Script

1. In JupyterLab, right-click an edited script and choose Set as Initialization Script from the shortcut menu to set the script as the initialization script for starting the instance.
2. If the message "Config Succeed" is displayed, the setting is successful.

**Figure 3-45 Successful setting**

```
Config succeed

The initialization script [/home/ma-user/work/install_package.sh] is successfully set up. It will be executed when the instance is started.
```

**Viewing and Updating the Initialization Script**

1. On the menu bar, choose **Settings > Advanced Setting Editor**. Alternatively, use the default shortcut key **Ctrl+** to view the script.
You can view the initialization script in the view on the right.

2. Edit the `scriptFile` content in **User Preferences** and click the save button in the upper right corner to update the initialization script.
Executing the Initialization Script

After the initialization script is set, the script is executed by default when the notebook instance is started.

- If the operation is successful, a success message is displayed in the lower right corner.
- If the execution fails, a dialog box is displayed. You can click View logs to view the logs.

Figure 3-49 Execution failed
4 Algorithm Management

4.1 Overview

ModelArts provides the algorithm management function to centrally manage your algorithms.

- **Creating an Algorithm**
  You can upload locally developed algorithms or algorithms developed using other tools to ModelArts for unified management. Then, you can use such algorithms to create training jobs on ModelArts and use the resources provided by ModelArts to obtain models.

- **Creating a Training Job Using an Algorithm**
  You can quickly create a training job on ModelArts to obtain the desired model using the algorithm you have created or subscribed to. You can quickly create a training job on ModelArts to obtain the desired model using the algorithm you have created.

4.2 Creating an Algorithm

If you have developed an algorithm locally, upload the algorithm to ModelArts for management. You can use cloud resources to train the uploaded algorithm to obtain a desired model. And then you can share the generated model to others through AI Gallery or deploy it as a service on ModelArts. For details about the algorithm format requirements, see [What Are the Format Requirements for Algorithms Imported from a Local Environment?](#).

**Creating an Algorithm**

1. Log in to the ModelArts management console and click **Algorithm Management** in the left navigation pane.
2. On the **My Algorithms** page, click **Create**. The **Create Algorithm** page is displayed.
3. On the **Create Algorithm** page, set related parameters.
   a. Set the basic parameters, including **Billing Mode**, **Name**, **Version**, and **Description**. For **Billing Mode**, only **Pay-per-use** is supported. Currently
no fee will generate because the algorithm management function is in the open beta test.

**Figure 4-1** Setting basic parameters

![Figure 4-1 Setting basic parameters](image)

b. Set parameter **Created by**. Currently, only custom scripts can be used to create algorithms. You need to set **AI Engine**, **Code Directory**, and **Boot File** based on the actual algorithm code. The framework of the AI engine you select must be the same as the one used when compiling algorithm code. For example, if TensorFlow is used for your algorithm code, select TensorFlow when you create an algorithm.

**Table 4-1** Parameter description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Engine</td>
<td>AI engine. For details about how to create an AI engine supported by an algorithm, see <strong>Table 5-27</strong>.</td>
</tr>
<tr>
<td>Code Directory</td>
<td>OBS path for storing algorithm code. Upload the code file to OBS in advance. The code directory cannot contain files or directories uploaded by others or irrelevant files or directories. Otherwise, the upload may fail.</td>
</tr>
<tr>
<td>Boot File</td>
<td>Must be in the code directory and end with <code>.py</code> or <code>.pyc</code>. That is, ModelArts supports only boot files compiled in Python.</td>
</tr>
</tbody>
</table>

**Figure 4-2** Creating an algorithm using custom scripts

![Figure 4-2 Creating an algorithm using custom scripts](image)

c. Set **Input Path Mapping Configuration** and **Output Path Mapping Configuration**. The two parameters are optional. You can determine whether to set them based on the site requirements. For details about the parameters, see **Table 4-2**.
The parameters configuration methods are as follows:

**Input Path Mapping Configuration:** Parameter `data_url` is displayed by default. You can enter the local path of the container in **Mapping Path**. You can enter only one directory name. Multi-level directories are not supported. That is, the slash (/) is not supported. Only letters, digits, hyphens (-), and underscores (_) are allowed. You can click **Add Input Path Mapping** to add a parameter, or click the deletion icon on the right to delete a parameter.

**Output Path Mapping Configuration:** Parameter `train_url` is displayed by default. You can enter the local path of the container in **Mapping Path**. You can enter only one directory name. Multi-level directories are not supported. That is, the slash (/) is not supported. Only letters, digits, hyphens (-), and underscores (_) are allowed. You can click **Add Output Path Mapping** to add a parameter, or click the deletion icon on the right to delete a parameter.

**Table 4-2 Parameter description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Detailed Description</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Path Mapping Configuration</td>
<td>Create an input path mapping for your algorithm. The mapping path is the local path of the computing container. When you use the algorithm to create a training task, a corresponding cloud storage location will be created for each input path mapping configuration. When the training starts, the system transmits the data in the cloud storage location to the corresponding mapping path (local path of the container). The local path of the container is sent to your algorithm script or image in the form of “key=value” command line parameter from the startup command. You only need to parse the input parameter and access the data in the local path of the container.</td>
<td>Simply speaking, the input mapping path is the local path of the container. It is used to copy OBS data to this local path for local code use when the training starts.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Detailed Description</td>
<td>Brief Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Output Path Mapping Configuration</td>
<td>Create an output path mapping for your algorithm. The mapping path is the local path of the computing container. When you use the algorithm to create a training task, a corresponding cloud storage location will be created for each output path mapping configuration. When the training completes, the system transmits the data in the mapping path (local path of the container) to the cloud storage location you select. The local path of the container is sent to your algorithm script or image in the form of “key=value” command line parameter from the startup command. You only need to parse the input parameter and save the output data in the local path of the container.</td>
<td>Simply speaking, the output mapping path is the local path of the container. It is used to copy the data generated during training to OBS after the training completes.</td>
</tr>
</tbody>
</table>

Figure 4-3 Setting Input Path Mapping Configuration and Output Path Mapping Configuration

- **Input Path Mapping Configuration**
  - Create an input path mapping for your algorithm. The mapping path is the local path of the computing container. When you use the algorithm to create a training task, a corresponding cloud storage location will be created for each input path mapping configuration. When the training starts, the system transmits the data in the cloud storage location to the corresponding mapping path (local path of the container). The local path of the container is sent to your algorithm script or image in the form of “key=value” command line parameter from the startup command. You only need to parse the input parameter and access the data in the local path of the container.

- **Output Path Mapping Configuration**
  - Create an output path mapping for your algorithm. The mapping path is the local path of the computing container. When you use the algorithm to create a training task, a corresponding cloud storage location will be created for each output path mapping configuration. When the training starts, the system transmits the data in the cloud storage location to the corresponding mapping path (local path of the container). The local path of the container is sent to your algorithm script or image in the form of “key=value” command line parameter from the startup command. You only need to parse the input parameter and access the data in the local path of the container.

- **Start command**
  - The start command is automatically generated and does not need to be entered.
e. Click **Next** in the lower right corner to switch to the **Hyperparameter Specifications** page.

f. On the **Hyperparameter Specifications** page, you can define hyperparameters of the algorithm based on your needs. The hyperparameter configurations are optional. You can skip it and proceed with the next step.

- **Import Parameters**: If you use a custom Python script to create an algorithm and use argparse or `tf.app.flags` in the boot file to define input parameter parsing, enable Import Parameters and set **Parameter Source** to the hyperparameter file path to pre-fill hyperparameters.

- **Show Example**: Click Show Example to view the example of hyperparameter configurations, from which you can learn how to set a hyperparameter. When the parameter type is string, the default value cannot be null.

- **Add Hyperparameter**: Click Add Hyperparameter below the list to add more hyperparameters. Set the hyperparameters based on Show Example.

- **User-Defined Hyperparameter**: Select **Enabled** or **Disabled** to determine whether to support custom hyperparameters.

You can delete, modify, or clear a hyperparameter by clicking corresponding buttons.

**Figure 4-5 Defining hyperparameters**

<table>
<thead>
<tr>
<th>Define Hyperparameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>When you use this algorithm to create a training job, you can view and modify the following hyperparameters, which will then be sent to your algorithm script or image as command line parameters from the boot command.</td>
</tr>
<tr>
<td>Delete</td>
</tr>
<tr>
<td>Import Parameters</td>
</tr>
</tbody>
</table>

If you use a custom Python script to create an algorithm and use argparse or `tf.app.flags` in the boot file to define input parameter parsing, the system will pre-fill hyperparameters for you. (The function is in the data list.)

<table>
<thead>
<tr>
<th>Parameter Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>/data/your_data/your_script/training_key.txt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Value Range</th>
<th>Adjustable</th>
<th>Default Value</th>
<th>Mandatory</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Show Example**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Value Range</th>
<th>Adjustable</th>
<th>Default Value</th>
<th>Mandatory</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Add Hyperparameter**

**User-Defined Hyperparameter**: Select **Enabled** or **Disabled**.
g. Click Next in the lower right corner to switch to the Constraints page.

h. On the Constraints page, you can define the input constraints and training constraints of the algorithm based on the site requirements.

i. Input constraints
   ○ **Data Source**: Set the value to a data storage location, ModelArts dataset, or both.
   ○ **Labeling Type**: Select the required labeling type from the drop-down list based on the algorithm if your data sources include ModelArts datasets.
   ○ **Data Format**: Select a data format if your data sources include ModelArts datasets. You can set the value to Default, CarbonData, or both. Default indicates the manifest format.
   ○ **Data Segmentation**: Select a data segmentation mode if your data sources include ModelArts datasets. Possible values are Segmented dataset, Dataset not segmented, and Unlimited.

   ![Algorithm input constraints](image)

ii. Training constraints
   ○ **Resource Type**: The options are CPU, GPU, and Ascend. You can select multiple options.
   ○ **Multicard Training**: Select Supported or Not supported.
   ○ **Distributed Training**: Select Supported or Not supported.
   ○ **Recommended Flavor**: Select the resource flavor recommended for the algorithm. That is, when you need to use your algorithm to train a model, you are advised to select the recommended resource flavor. In the drop-down list, you can select desired resource flavors supported by ModelArts for training.
Follow-Up Procedure

You can use the created algorithm to quickly create a training job and build a model. For details, see Creating a Training Job Using an Algorithm.

4.3 Creating a Training Job Using an Algorithm

You can use a created algorithm or an algorithm subscribed from AI Gallery to quickly create a training job and build a model.

Creating a Training Job

The algorithms you create are displayed on the Algorithm Management > My Algorithms page, and the algorithms you subscribe to are displayed on the Algorithm Management > My Subscriptions page. The two types of algorithms can be used to quickly create a training job. You can use either of the following ways to create a training job using an available algorithm:

- Choose Algorithm Management > My Algorithms, select a desired algorithm, and click Create Training Job in the Operation column to create a training job using the algorithm. Using Existing Algorithms to Train Models describes the parameters.
- Choose Algorithm Management > My Subscriptions, click the triangle on the left of the item name to expand the details. In the Versions area, click Create Training Job to create a training job using the algorithm. Using Existing Algorithms to Train Models describes the parameters.
- Choose Training Management > Training Jobs, click Create, and select Algorithm Management for Algorithm Source. Then, you can select an
existing algorithm to create a training job. Using Existing Algorithms to Train Models describes the parameters.
5 Training Management

5.1 Model Training Overview

ModelArts provides model training for you to view the training effect, based on which you can adjust your model parameters. You can select resource pools (CPU or GPU) with different instance flavors for model training. In addition to the models developed by users, ModelArts also provides built-in algorithms. You can directly adjust parameters of the built-in algorithms, instead of developing a model by yourself, to obtain a satisfactory model.

Description of the Model Training Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-in algorithms</td>
<td>Based on the frequently-used AI engines in the industry, ModelArts provides built-in algorithms to meet a wide range of your requirements. You can directly select the algorithms for training jobs, without concerning model development.</td>
<td>Introduction to Built-in Algorithms</td>
</tr>
<tr>
<td>Training job management</td>
<td>You can create training jobs, manage training job versions, and view details of training jobs, and evaluation details.</td>
<td>Introduction Managing Training Job Versions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Viewing Job Details</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Reference</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Job parameter management</td>
<td>You can save the parameter settings of a training job (including the data source, algorithm source, running parameters, resource pool parameters, and more) as a job parameter, which can be directly used when you create a training job, eliminating the need to set parameters one by one. As such, the configuration efficiency can be greatly improved.</td>
<td>Managing Job Parameters</td>
</tr>
<tr>
<td>Model training visualization</td>
<td>TensorBoard is a tool that can effectively display the computational graph of TensorFlow in the running process, the trend of various metrics in time, and the data used in the training. Currently, TensorBoard supports only the training jobs based on the TensorFlow and MXNet engines.</td>
<td>Managing Visualization Jobs</td>
</tr>
</tbody>
</table>

### 5.2 Built-in Algorithms

#### 5.2.1 Introduction to Built-in Algorithms

Based on the frequently-used AI engines in the industry, ModelArts provides built-in algorithms to meet a wide range of your requirements. You can directly select the algorithms for training jobs, without concerning model development.

Built-in algorithms of ModelArts adopt MXNet and TensorFlow engines and are mainly used for detection of object classes and locations, image classification, semantic image segmentation, and reinforcement learning.

#### Viewing Built-in Algorithms

In the left navigation pane of the ModelArts management console, choose **Training Management > Training Jobs**. On the displayed page, click **Built-in Algorithms**. In the built-in algorithm list, click ✓ next to an algorithm name to view details about the algorithm.

You can click **Create Training Job** in the **Operation** column for an algorithm to quickly create a training job, for which this algorithm serves as the **Algorithm Source**.

**NOTE**

- Before using a built-in algorithm to create a training job, prepare and upload training data to OBS. For details about the data storage path and data format requirements, see **Requirements on Datasets**.
- The built-in algorithms hard_example_mining and feature_cluster are for internal use only and you cannot use them for training.
For details about the built-in algorithms and their running parameters, see the following:

- yolo_v3
- yolov3_resnet18
- retinanet_resnet_v1_50
- inception_v3
- darknet_53
- SegNet_VGG_BN_16
- ResNet_v2_50
- ResNet_v1_50
- Faster_RCNN_ResNet_v2_101
- Faster_RCNN_ResNet_v1_50

5.2.2 Requirements on Datasets

The built-in algorithms provided by ModelArts can be used for image classification, object detection, image semantic segmentation, and reinforcement learning. The requirements for the datasets vary according to the built-in algorithms used for different purposes. Before using a built-in algorithm to create a training job, you are advised to prepare a dataset based on the requirements of the algorithm.

Image Classification

The training dataset must be stored in the OBS bucket. The following shows the OBS path structure of the dataset:

|-- data_url
    |--a.jpg
    |--a.txt
    |--b.jpg
    |--b.txt
    ...

- **data_url** indicates the folder name. You can customize the folder name. Images and label files cannot be stored in the root directory of an OBS bucket.

- Images and label files must have the same name. The .txt files are label files for image classification. The images can be in JPG, JPEG, PNG, or BMP format.

- The first row of label files for image classification indicates the category name of images, which can be Chinese characters, English letters, or digits. The following provides an example of file content:

  cat

- In addition to the preceding files and folders, no other files or folders can exist in the **data_url** folder.

- You can directly use an existing image classification dataset with published versions in **Data Management** of ModelArts.

- You can also name sub-folders in the **data_url** directory by label, as shown in the following:

  |-- data_url
     |--cat
     |--a.jpg
Object Detection and Locating

The training dataset must be stored in the OBS bucket. The following shows the OBS path structure of the dataset:

```
|-- data_url
  |-- a.jpg
  |-- a.xml
  |-- b.jpg
  |-- b.xml
  ...
```

- **data_url** indicates the folder name. You can customize the folder name. Images and label files cannot be stored in the root directory of an OBS bucket.
- Images and label files must have the same name. The .xml files are label files for object detection. The images can be in JPG, JPEG, PNG, or BMP format.
- In addition to the preceding files and folders, no other files or folders can exist in the `data_url` folder.
- You can directly use an existing object detection dataset with published versions in Data Management of ModelArts.
- The following provides a label file for object detection. The key parameters are size (image size), object (object information), and name (label name, which can be Chinese characters, English letters, or digits). Note that the values of xmin, ymin, xmax, and ymax in the bndbox field cannot exceed the value of size. That is, the value of min cannot be less than 0, and the value of max cannot be greater than the value of width or height.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<annotation>
  <folder>Images</folder>
  <filename>IMG_20180919_120022.jpg</filename>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>800</width>
    <height>600</height>
    <depth>1</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>yunbao</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>216.00</xmin>
      <ymin>108.00</ymin>
      <xmax>705.00</xmax>
      <ymax>488.00</ymax>
    </bndbox>
  </object>
</annotation>
```
Image Semantic Segmentation

The training dataset must be stored in the OBS bucket. The following shows the OBS path structure of the dataset:

|-- data_url
   |--Image
       |--a.jpg
       |--b.jpg
       ...
   |--Label
       |--a.jpg
       |--b.jpg
       ...
   |--train.txt
   |--val.txt

Description:

- **data_url**, **Image**, and **Label** indicate the OBS folder names. The **Image** folder stores images for semantic segmentation, and the **Label** folder stores labeled images.

- The name and format of the images for semantic segmentation must be the same as those of the corresponding labeled images. Images in JPG, JPEG, PNG, and BMP formats are supported.

- In the preceding code snippet, **train.txt** and **val.txt** are two list files. **train.txt** is the list file of the training set, and **val.txt** is the list file of the validation set. It is recommended that the ratio of the training set to the validation set be 8:2.

   In the list file, the relative paths of images and labels are separated by spaces. Different pieces of data are separated by newline characters. The following gives an example:

   Image/a.jpg Label/a.jpg
   Image/b.jpg Label/b.jpg
   ...

5.2.3 Algorithms and Their Running Parameters

This section describes the built-in algorithms supported by ModelArts and the running parameters supported by each algorithm. You can set running parameters for a training job as required.

**yolo_v3**

<table>
<thead>
<tr>
<th>Table 5-2 Algorithm description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Usage</td>
</tr>
<tr>
<td>Engine Type</td>
</tr>
</tbody>
</table>
**Parameter** | **Description**
--- | ---
Precision | 81.7% (mAP)  
**mAP** is an indicator that measures the effect of an object detection algorithm. For object detection tasks, the precision rate (Precision) and recall rate (Recall) can be calculated for each class of object. The rates can be calculated and tested multiple times for each class of object based on different thresholds, and a P-R curve is obtained accordingly. The area under the curve is the average value.

Training Dataset | PASCAL VOC2007, detection of 20 classes of objects

Data Format | shape: \[H\geq224, W\geq224, C\geq1\]; type: int8

Running Parameter | lr=0.0001 ; mom=0.9 ; wd=0.0005  
For more available running parameters, see **Table 5-3**.

---

**Table 5-3 Running parameters**

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Parameter Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lr</td>
<td>Learning rate</td>
<td>0.0001</td>
</tr>
<tr>
<td>mom</td>
<td>Momentum of the training network</td>
<td>0.9</td>
</tr>
<tr>
<td>wd</td>
<td>Parameter weight decay coefficient, L2</td>
<td>0.0005</td>
</tr>
<tr>
<td>num_classes</td>
<td>Total number of image classes used in training. If you add images of other classes, this parameter will be automatically identified and matched. You do not need to manually change the parameter value.</td>
<td>None</td>
</tr>
<tr>
<td>split_spec</td>
<td>Split ratio of the training set and validation set</td>
<td>0.8</td>
</tr>
<tr>
<td>batch_size</td>
<td>Total number of training images updated each time</td>
<td>16</td>
</tr>
<tr>
<td>eval_frecuence</td>
<td>Frequency for validating the model. By default, validation is performed every epoch.</td>
<td>1</td>
</tr>
<tr>
<td>num_epoch</td>
<td>Number of training epochs</td>
<td>10</td>
</tr>
</tbody>
</table>
Table 5-4 Running parameters that can be added

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Parameter Description</th>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_examples</td>
<td>Total number of images used for training. For example, if the total number of images is 1,000, the images used for training is 800 based on the split ratio.</td>
<td>16551</td>
</tr>
<tr>
<td>disp_batches</td>
<td>The loss and training speed of the model is displayed every $N$ batches.</td>
<td>20</td>
</tr>
<tr>
<td>warm_up_epochs</td>
<td>Number of epochs when the target learning rate of the warm-up strategy is reached</td>
<td>0</td>
</tr>
<tr>
<td>lr_steps</td>
<td>Number of epochs when the learning rate attenuates in the multi-factor strategy. By default, the learning rate attenuates to 0.1 times of the original value at the 10th and 15th epochs.</td>
<td>10,15</td>
</tr>
</tbody>
</table>

**yolov3_resnet18**

**NOTE**

This algorithm supports inference based on Ascend 310 but does not support CPU- or GPU-based inference. If CPU- or GPU-based inference is required, you are advised to use the algorithm developed by the MXNet engine, for example, **yolo_v3**. The two algorithms are used for the same purpose. The yolo_v3 algorithm is applicable to CPU- or GPU-based inference, and the yolov3_resnet18 algorithm is applicable to inference based on Ascend 310.

Table 5-5 Algorithm description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>yolov3_resnet18</td>
</tr>
<tr>
<td>Usage</td>
<td>Object detection and locating</td>
</tr>
<tr>
<td>Engine Type</td>
<td>TensorFlow, TF-1.13.1-python3.6</td>
</tr>
<tr>
<td>Precision</td>
<td>-</td>
</tr>
<tr>
<td>Training Dataset</td>
<td>COCO</td>
</tr>
<tr>
<td>Data Format</td>
<td>shape: [H&gt;=224, W&gt;=224, C&gt;=1]; type: int8</td>
</tr>
<tr>
<td>Running Parameter</td>
<td>input_shape=352,640; batch_size=32; max_epochs=100; ...</td>
</tr>
</tbody>
</table>

For more available running parameters, see **Table 5-6**.
### Table 5-6 Running parameters

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Parameter Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_shape</td>
<td>Before training, the algorithm adjusts the height and width of the input image. The input format is <code>height,width</code>. No space is allowed between commas (,).</td>
<td>352,640</td>
</tr>
<tr>
<td>max_epochs</td>
<td>Maximum number of iterations during model running. The value must be an integer.</td>
<td>32</td>
</tr>
<tr>
<td>batch_size</td>
<td>Number of images for each iteration (standalone). The value must be an integer.</td>
<td>100</td>
</tr>
<tr>
<td>learning_rate</td>
<td>Learning rate during model running. The value ranges from 0 to 1.</td>
<td>0.001</td>
</tr>
<tr>
<td>obj_threshold</td>
<td>Confidence score. The value ranges from 0 to 1. During inference, if the confidence score of a bounding box is less than the value of this parameter, the box is filtered out.</td>
<td>0.3</td>
</tr>
<tr>
<td>nms_threshold</td>
<td>Non-maximum suppression (NMS) threshold. The value ranges from 0 to 1.</td>
<td>0.4</td>
</tr>
<tr>
<td>class_names</td>
<td>If <code>Data Source</code> is set to <code>Dataset</code>, you do not need to set this parameter. If <code>Data Source</code> is set to <code>Data path</code>, you must manually add this parameter. Labels of the dataset. Use commas (,) to separate multiple labels and no space is allowed. For example, <code>person,face</code>.</td>
<td>-</td>
</tr>
</tbody>
</table>

---

**retinanet_resnet_v1_50**

### Table 5-7 Algorithm description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>retinanet_resnet_v1_50</td>
</tr>
<tr>
<td>Usage</td>
<td>Object detection and locating</td>
</tr>
<tr>
<td>Engine Type</td>
<td>TensorFlow, TF-1.8.0-python2.7</td>
</tr>
</tbody>
</table>
### Table 5-8 Running parameters

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Parameter Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>split_spec</td>
<td>Split ratio of the training set and validation set</td>
<td>train:0.8, eval: 0.2</td>
</tr>
<tr>
<td>num_gpus</td>
<td>Number of used GPUs</td>
<td>1</td>
</tr>
<tr>
<td>batch_size</td>
<td>Number of images for each iteration (standalone). To ensure the algorithm precision, you are advised to use the default value.</td>
<td>The value is fixed to 1.</td>
</tr>
<tr>
<td>learning_rate_strategy</td>
<td>Learning rate strategy. The value ranges from 0 to 1. For example, the value can be set to 0.001.</td>
<td>0.002</td>
</tr>
<tr>
<td>evaluate_every_n_epochs</td>
<td>A validation is performed after N epochs are trained.</td>
<td>1</td>
</tr>
<tr>
<td>save_interval_secs</td>
<td>Interval for saving the model. The unit is second. If model running time is greater than 2,000,000s, the model is saved once every 2,000,000s by default. If model running time is less than 2,000,000s, the model is saved when the running is complete.</td>
<td>2000000</td>
</tr>
<tr>
<td>max_epochs</td>
<td>Maximum number of training epochs</td>
<td>100</td>
</tr>
<tr>
<td>Optional Parameter</td>
<td>Parameter Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>log_every_n_steps</td>
<td>Logs are printed every $N$ steps. By default, logs are printed every 10 steps.</td>
<td>10</td>
</tr>
<tr>
<td>save_summaries_steps</td>
<td>Summary information is saved every five steps, including the model gradient update value and training parameters.</td>
<td>5</td>
</tr>
<tr>
<td>weight_decay</td>
<td>L2 regularization weight decay</td>
<td>0.00004</td>
</tr>
<tr>
<td>optimizer</td>
<td>Optimizer. The options are as follows:</td>
<td>momentum</td>
</tr>
<tr>
<td>momentum</td>
<td>Optimizer parameter momentum</td>
<td>0.9</td>
</tr>
<tr>
<td>patience</td>
<td>After training of $N$ epochs, if the precision (mAP for object detection and accuracy for image classification) does not increase compared with the previous maximum value, that is, the difference between the precision and the maximum precision is less than the value of $\text{decay_min_delta}$, the learning rate attenuates to one tenth of the original value. The default value of $N$ is 8.</td>
<td>8</td>
</tr>
<tr>
<td>decay_patience</td>
<td>After training of extra $M$ epochs on the basis of the preceding patience, if the precision (mAP for object detection and accuracy for image classification) does not increase, that is, the difference between the precision and the maximum precision is less than the value of $\text{decay_min_delta}$, training will be terminated early. The default value of $M$ is 1.</td>
<td>1</td>
</tr>
<tr>
<td>decay_min_delta</td>
<td>Minimum difference between the precision (mAP for object detection and accuracy for image classification) corresponding to different learning rates. If the parameter value is greater than 0.001, the precision is increased. Otherwise, the precision is not increased.</td>
<td>0.001</td>
</tr>
<tr>
<td>rcnn_iou_threshold</td>
<td>IoU threshold used for calculating the map when SSD or Faster R-CNN are used</td>
<td>0.5</td>
</tr>
</tbody>
</table>
### inception_v3

**Table 5-9 Algorithm description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>inception_v3</td>
</tr>
<tr>
<td>Usage</td>
<td>Image Classification</td>
</tr>
<tr>
<td>Engine Type</td>
<td>TensorFlow, TF-1.8.0-python2.7</td>
</tr>
<tr>
<td>Precision</td>
<td>78.00%(top1), 93.90%(top5)</td>
</tr>
<tr>
<td></td>
<td>- <strong>top1</strong> indicates that the classification is considered correct only when the image with the maximum probability is the correct image.</td>
</tr>
<tr>
<td></td>
<td>- <strong>top5</strong> indicates that the classification is considered correct only when the correct image is within the top 5 images.</td>
</tr>
<tr>
<td>Training Dataset</td>
<td>ImageNet, classification of 1,000 image classes</td>
</tr>
<tr>
<td>Data Format</td>
<td>shape: [H, W, C&gt;=1]; type: int8</td>
</tr>
<tr>
<td>Running Parameter</td>
<td>batch_size=32 ; split_spec=train:0.8,eval:0.2 ;</td>
</tr>
<tr>
<td></td>
<td>For more available running parameters, see Table 5-10.</td>
</tr>
</tbody>
</table>

**Table 5-10 Running parameters**

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Parameter Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>split_spec</td>
<td>Split ratio of the training set and validation set</td>
<td>train:0.8,eval:0.2</td>
</tr>
<tr>
<td>num_gpus</td>
<td>Number of used GPUs</td>
<td>1</td>
</tr>
<tr>
<td>batch_size</td>
<td>Number of images for each iteration (standalone). To ensure the algorithm precision, you are advised to use the default value.</td>
<td>32</td>
</tr>
<tr>
<td>eval_batch_size</td>
<td>Number of images read each step during validation (standalone)</td>
<td>32</td>
</tr>
<tr>
<td>learning_rate_strategy</td>
<td>Learning rate strategy. For example, 10:0.001,20:0.0001 indicates that the learning rate for 0 to 10 epochs is 0.001, and that for 10 to 20 epochs is 0.0001.</td>
<td>0.002</td>
</tr>
<tr>
<td>evaluate_every_n_epochs</td>
<td>A validation is performed after N epochs are trained.</td>
<td>1</td>
</tr>
</tbody>
</table>
### Table 5-11 Running parameters that can be added

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Parameter Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight_decay</td>
<td>L2 regularization weight decay</td>
<td>0.00004</td>
</tr>
<tr>
<td>optimizer</td>
<td>Optimizer. The options are as follows:</td>
<td>momentum</td>
</tr>
<tr>
<td></td>
<td>● dymomentumw</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● sgd</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● adam</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● momentum</td>
<td></td>
</tr>
<tr>
<td>momentum</td>
<td>Optimizer parameter momentum</td>
<td>0.9</td>
</tr>
<tr>
<td>patience</td>
<td>After training of $N$ epochs, if the precision (mAP for object detection and accuracy for image classification) does not increase compared with the previous maximum value, that is, the difference between the precision and the maximum precision is less than the value of decay_min_delta, the learning rate attenuates to one tenth of the original value. The default value of $N$ is 8.</td>
<td>8</td>
</tr>
<tr>
<td>Optional Parameter</td>
<td>Parameter Description</td>
<td>Recommend ed Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>decay_patience</td>
<td>After training of extra $M$ epochs on the basis of the preceding patience, if the precision (mAP for object detection and accuracy for image classification) does not increase, that is, the difference between the precision and the maximum precision is less than the value of <code>decay_min_delta</code>, training will be terminated early. The default value of $M$ is 1.</td>
<td>1</td>
</tr>
<tr>
<td>decay_min_delta</td>
<td>Minimum difference between the precision (mAP for object detection and accuracy for image classification) corresponding to different learning rates. If the parameter value is greater than 0.001, the precision is increased. Otherwise, the precision is not increased.</td>
<td>0.001</td>
</tr>
<tr>
<td>image_size</td>
<td>Size of the input image. If this parameter is set to <code>None</code>, the default image size prevails.</td>
<td>None</td>
</tr>
<tr>
<td>lr_warmup_strategy</td>
<td>Warm-up strategy (linear or exponential)</td>
<td>linear</td>
</tr>
<tr>
<td>num_readers</td>
<td>Number of threads for reading data</td>
<td>64</td>
</tr>
<tr>
<td>fp16</td>
<td>Whether to use FP16 for training</td>
<td>FALSE</td>
</tr>
<tr>
<td>max_lr</td>
<td>Maximum learning rate for the <code>dymomentum</code> and <code>dymomentumw</code> optimizers, or when <code>use_lr_schedule</code> is used.</td>
<td>6.4</td>
</tr>
<tr>
<td>min_lr</td>
<td>Minimum learning rate for the <code>dymomentum</code> and <code>dymomentumw</code> optimizers, or when <code>use_lr_schedule</code> is used.</td>
<td>0.005</td>
</tr>
<tr>
<td>warmup</td>
<td>Proportion of warm-up in total training steps. This parameter is valid when <code>use_lr_schedule</code> is <code>lcd</code> or <code>poly</code>.</td>
<td>0.1</td>
</tr>
<tr>
<td>cooldown</td>
<td>Minimum learning rate in the warm-up</td>
<td>0.05</td>
</tr>
<tr>
<td>max_mom</td>
<td>Maximum momentum. This parameter is valid for dynamic momentum.</td>
<td>0.98</td>
</tr>
<tr>
<td>min_mom</td>
<td>Minimum momentum. This parameter is valid for dynamic momentum.</td>
<td>0.85</td>
</tr>
<tr>
<td>use_lars</td>
<td>Whether to use LARS</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
### Table 5-12 Algorithm description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>darknet_53</td>
</tr>
<tr>
<td>Usage</td>
<td>Image Classification</td>
</tr>
<tr>
<td>Engine Type</td>
<td>MXNet, MXNet-1.2.1-python2.7</td>
</tr>
<tr>
<td>Precision</td>
<td>78.56%(top1), 94.43%(top5)</td>
</tr>
<tr>
<td></td>
<td>• <strong>top1</strong> indicates that the classification is</td>
</tr>
<tr>
<td></td>
<td>considered correct only when the image with the</td>
</tr>
<tr>
<td></td>
<td>maximum probability is the correct image.</td>
</tr>
<tr>
<td></td>
<td>• <strong>top5</strong> indicates that the classification is</td>
</tr>
<tr>
<td></td>
<td>considered correct only when the correct image</td>
</tr>
<tr>
<td></td>
<td>is within the top 5 images.</td>
</tr>
<tr>
<td>Training Dataset</td>
<td>ImageNet, classification of 1,000 image classes</td>
</tr>
<tr>
<td>Data Format</td>
<td>shape: ([H&gt;=224, W&gt;=224, C&gt;=1]); type: int8</td>
</tr>
<tr>
<td>Running Parameter</td>
<td>split_spec=0.8 ; batch_size=4 ;</td>
</tr>
<tr>
<td></td>
<td>For more available running parameters, see Table 5-13.</td>
</tr>
</tbody>
</table>

### Table 5-13 Running parameters

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Parameter Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>split_spec</td>
<td>Split ratio of the training set and validation set</td>
<td>0.8</td>
</tr>
<tr>
<td>batch_size</td>
<td>Total amount of input data each time the parameters are updated</td>
<td>4</td>
</tr>
<tr>
<td>lr</td>
<td>Learning rate of the updated parameters</td>
<td>0.0001</td>
</tr>
</tbody>
</table>
Table 5-14 Algorithm description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>SegNet_VGG_BN_16</td>
</tr>
<tr>
<td>Usage</td>
<td>Image semantic segmentation</td>
</tr>
<tr>
<td>Engine Type</td>
<td>MXNet, MXNet-1.2.1-python2.7</td>
</tr>
<tr>
<td>Precision</td>
<td>89% (pixel acc)</td>
</tr>
<tr>
<td></td>
<td>pixel acc indicates the ratio of correct pixels to total pixels.</td>
</tr>
<tr>
<td>Training Dataset</td>
<td>Camvid</td>
</tr>
<tr>
<td>Data Format</td>
<td>shape: [H=360, W=480, C==3]; type: int8</td>
</tr>
<tr>
<td>Running Parameter</td>
<td>deploy_on_terminal=False;</td>
</tr>
<tr>
<td></td>
<td>For more available running parameters, see Table 5-15.</td>
</tr>
</tbody>
</table>

Table 5-15 Running parameters

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Parameter Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lr</td>
<td>Learning rate of the updated parameters</td>
<td>0.0001</td>
</tr>
<tr>
<td>mom</td>
<td>Momentum of the training network</td>
<td>0.9</td>
</tr>
<tr>
<td>wd</td>
<td>Attenuation coefficient</td>
<td>0.0005</td>
</tr>
<tr>
<td>num_classes</td>
<td>Total number of image classes in training. You do not need to plus 1 here.</td>
<td>11</td>
</tr>
<tr>
<td>batch_size</td>
<td>Total number of training images updated each time</td>
<td>8</td>
</tr>
<tr>
<td>num_epoch</td>
<td>Number of training epochs</td>
<td>15</td>
</tr>
</tbody>
</table>

SegNet_VGG_BN_16
### Optional Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>save_frequency</td>
<td>Interval for saving the model, indicating that the model is saved every $N$ epochs</td>
<td>1</td>
</tr>
<tr>
<td>num_examples</td>
<td>Total number of images used for training, which indicates the number of files in <code>train.txt</code></td>
<td>2953</td>
</tr>
</tbody>
</table>

**ResNet_v2_50**

**Table 5-16 Algorithm description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ResNet_v2_50</td>
</tr>
<tr>
<td>Usage</td>
<td>Image Classification</td>
</tr>
<tr>
<td>Engine Type</td>
<td>MXNet, MXNet-1.2.1-python2.7</td>
</tr>
<tr>
<td>Precision</td>
<td>75.55%(top1), 92.6%(top5)</td>
</tr>
<tr>
<td></td>
<td>• <strong>top1</strong> indicates that the classification is considered correct only when the image with the maximum probability is the correct image.</td>
</tr>
<tr>
<td></td>
<td>• <strong>top5</strong> indicates that the classification is considered correct only when the correct image is within the top 5 images.</td>
</tr>
<tr>
<td>Training Dataset</td>
<td>ImageNet, classification of 1,000 image classes</td>
</tr>
<tr>
<td>Data Format</td>
<td>shape: [H&gt;=32, W&gt;=32, C&gt;=1]; type: int8</td>
</tr>
<tr>
<td>Running Parameter</td>
<td>split_spec=0.8 ; batch_size=4 ; The available running parameters are the same as those for the darknet_53 algorithm. For details, see Table 5-13.</td>
</tr>
</tbody>
</table>

**ResNet_v1_50**

**Table 5-17 Algorithm description**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ResNet_v1_50</td>
</tr>
<tr>
<td>Usage</td>
<td>Image Classification</td>
</tr>
<tr>
<td>Engine Type</td>
<td>TensorFlow, TF-1.8.0-python2.7</td>
</tr>
</tbody>
</table>
**Parameter** | **Description**
---|---
Precision | 74.2%(top1), 91.7%(top5)
  * **top1** indicates that the classification is considered correct only when the image with the maximum probability is the correct image.
  * **top5** indicates that the classification is considered correct only when the correct image is within the top 5 images.

**Training Dataset** | ImageNet, classification of 1,000 image classes
**Data Format** | shape: \([H>=600, W<=1024, C>=1]\); type: int8
**Running Parameter** | batch_size=32 ; split_spec=train:0.8,eval:0.2 ;
The available running parameters are the same as those for the **inception_v3** algorithm. For details, see **Table 5-10**.

**Faster_RCNN_ResNet_v2_101**

To achieve a satisfactory training effect, fine-tuning of parameters is required when this algorithm is used. Otherwise, the result may not meet the expectation. Alternatively, you can use other algorithms.

**Table 5-18** Algorithm description

<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Faster_RCNN_ResNet_v2_101</td>
</tr>
<tr>
<td>Usage</td>
<td>Object detection and locating</td>
</tr>
<tr>
<td>Engine Type</td>
<td>MXNet, MXNet-1.2.1-python2.7</td>
</tr>
<tr>
<td>Precision</td>
<td>80.05%(mAP)</td>
</tr>
</tbody>
</table>

**mAP** is an indicator that measures the effect of an object detection algorithm. For object detection tasks, the precision rate (Precision) and recall rate (Recall) can be calculated for each class of object. The rates can be calculated and tested multiple times for each class of object based on different thresholds, and a P-R curve is obtained accordingly. The area under the curve is the average value.

<table>
<thead>
<tr>
<th>Training Dataset</th>
<th>PASCAL VOC2007, detection of 20 classes of objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Format</td>
<td>shape: ([H, W, C==3]); type: int8</td>
</tr>
<tr>
<td>Running Parameter</td>
<td>lr=0.0001 ; eval_frequency=1 ;</td>
</tr>
</tbody>
</table>
For more available running parameters, see **Table 5-19**.
Table 5-19 Running parameters

<table>
<thead>
<tr>
<th>Optional Parameter</th>
<th>Parameter Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_classes</td>
<td>Total number of image classes in training. The value must plus 1 because there is a background class.</td>
<td>None</td>
</tr>
<tr>
<td>eval_frequency</td>
<td>Frequency for validating the model. By default, validation is performed every epoch.</td>
<td>1</td>
</tr>
<tr>
<td>lr</td>
<td>Learning rate</td>
<td>0.0001</td>
</tr>
<tr>
<td>mom</td>
<td>Momentum of the training network</td>
<td>0.9</td>
</tr>
<tr>
<td>wd</td>
<td>Parameter weight decay coefficient, L2</td>
<td>0.0005</td>
</tr>
<tr>
<td>split_spec</td>
<td>Split ratio of the training set and validation set</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Faster_RCNN_ResNet_v1_50

Table 5-20 Algorithm description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Faster_RCNN_ResNet_v1_50</td>
</tr>
<tr>
<td>Usage</td>
<td>Object detection and locating</td>
</tr>
<tr>
<td>Engine Type</td>
<td>TensorFlow, TF-1.8.0-python2.7</td>
</tr>
<tr>
<td>Precision</td>
<td>73.6%(mAP)</td>
</tr>
<tr>
<td>mAP is an indicator that measures the effect of an object detection algorithm. For object detection tasks, the precision rate (Precision) and recall rate (Recall) can be calculated for each class of object. The rates can be calculated and tested multiple times for each class of object based on different thresholds, and a P-R curve is obtained accordingly. The area under the curve is the average value.</td>
<td></td>
</tr>
<tr>
<td>Training Dataset</td>
<td>PASCAL VOC2007, detection of 20 classes of objects</td>
</tr>
<tr>
<td>Data Format</td>
<td>shape: [H&gt;=600,W&lt;=1024,C&gt;=1];type:int8</td>
</tr>
<tr>
<td>Running Parameter</td>
<td>Such as split_spec=train:0.8,eval:0.2, num_gpus=1, batch_size=32, and eval_batch_size=32. The default running parameters are the same as those of the retinanet_resnet_v1_50 algorithm. For details about the parameters and default values, see Table 5-8.</td>
</tr>
</tbody>
</table>
5.3 Creating a Training Job

5.3.1 Introduction

ModelArts supports multiple types of training jobs during the entire AI development process. Select a creation mode based on the algorithm source.

Algorithm Sources of Training Jobs

- **Built-in**
  If you do not know how to develop algorithms, you can use the built-in algorithms of ModelArts. After simple parameters adjustment, you can create a training job and build a model. For details about the operation guide, see Using Built-in Algorithms to Train Models. For details about the existing built-in algorithms and running parameters, see Introduction to Built-in Algorithms.

- **Frequently-used**
  If you have used some frequently-used frameworks to develop algorithms locally, you can select a frequently-used framework and create a training job to build a model. For details, see Using Frequently-used Frameworks to Train Models.

- **Custom**
  If the framework used for algorithm development is not a frequently-used framework, you can build an algorithm into a custom image and use the custom image to create a training job. For details about the operation guide to create a training job, see Using Custom Images to Train Models. For details about the specifications and description of custom images, see Specifications for Custom Images Used for Training Jobs.

5.3.2 Using Existing Algorithms to Train Models

You can quickly use a created algorithm to create a training job and build a model.

Prerequisites

- Data has been prepared. Specifically, you have created an available dataset in ModelArts, or you have uploaded the dataset used for training to the OBS directory.
- In Algorithm Management, you have created an algorithm.
- At least one empty folder has been created on OBS for storing the training output.
- The account is not in arrears because resources are consumed when training jobs are running.
- The OBS directory you use and ModelArts are in the same region.
Precautions

In the dataset directory specified for a training job, the names of the files (such as the image file, audio file, and label file) containing data used for training contain 0 to 255 characters. If the names of certain files in the dataset directory contain over 255 characters, the training job will ignore these files and use data in the valid files for training. If the names of all files in the dataset directory contain over 255 characters, no data is available for the training job and the training job fails.

Creating a Training Job

1. Log in to the ModelArts management console. In the left navigation pane, choose Training Management > Training Jobs. By default, the system switches to the Training Jobs page.
2. In the upper left corner of the training job list, click Create to switch to the Create Training Job page.
3. Set related parameters and click Next.
   a. Set the basic information, including Billing Mode, Name, Version, and Description. For Billing Mode, only Pay-per-use is supported, and the mode cannot be changed. For Version, the system automatically creates a version number, which is named according to a certain rule, for example, V001 and V002. The version number cannot be changed. Specify Name and Description according to actual requirements.

   Figure 5-1 Setting basic information about the training job

   ![Billing Mode](image)
   ![Name](image)
   ![Version](image)
   ![Description](image)

   b. Set job parameters, including the data source, algorithm source, and more. For details, see Table 5-21. The value range of the data source is consistent with the constraints of existing algorithms.

   Figure 5-2 Algorithm Management as the algorithm source

   ![Algorithm Source](image)
Table 5-21 Job parameter description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sub-Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm Source</td>
<td>Algorithm Management</td>
<td>Select Algorithm Management and click Select on the right of the algorithm name. The Algorithm Management page is displayed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- On the My Algorithms tab page, you can select a created algorithm. For details about how to create an algorithm, see Creating an Algorithm.</td>
</tr>
<tr>
<td>Training Input</td>
<td>Data Source &gt; Dataset</td>
<td>Select an available dataset and its version from the ModelArts Data Management module.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Dataset</strong>: Select a published dataset from the drop-down list. If no dataset is available in ModelArts, no result will be displayed in the drop-down list.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Version</strong>: Select a version based on the selected dataset.</td>
</tr>
<tr>
<td></td>
<td>Data Source &gt; Data path</td>
<td>Select the training data from your OBS bucket.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>On the right of the Data path text box, click Select. In the dialog box that is displayed, select an OBS folder for storing data.</td>
</tr>
<tr>
<td>Training Output</td>
<td>Model Output</td>
<td>Select the storage path of the training result (OBS path). To avoid errors, you are advised to select an empty directory for Model Output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do not use the dataset storage directory as the training output location.</td>
</tr>
<tr>
<td>Tuning Parameter</td>
<td>-</td>
<td>The value of this parameter varies according to the selected algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If tuning parameters are defined for the created or subscribed algorithm, you need to set the parameters when creating a training job. You can click Add Tuning Parameter to add multiple tuning parameters.</td>
</tr>
<tr>
<td>Job Log Path</td>
<td>-</td>
<td>Select a path for storing log files generated during job running.</td>
</tr>
</tbody>
</table>

C. Select resources for the training job. The value range of the training parameters is consistent with the constraints of existing algorithms.
Figure 5-3 Selecting resources for the training job

Table 5-22 Resource parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Pool</td>
<td>Select resource pools for the job. For training jobs, <strong>Public resource pools</strong> and <strong>Dedicated resource pools</strong> are available. Currently, <strong>Dedicated resource pools</strong> are not supported.</td>
</tr>
<tr>
<td>Instance Flavor</td>
<td>Select a resource flavor based on the resource type. The GPU resource delivers better performance, and the CPU resource is more cost effective. If your algorithm has been defined to use CPUs or GPUs, you can select a proper resource flavor based on the constraints of existing algorithms. Invalid options are grayed out. Currently, free-of-charge flavors are not supported when you use an existing algorithm to create a training job. The data disk capacity varies depending on the resource type. For details, see <a href="#">What Are Sizes of the /cache Directories for GPU and CPU Resources in the Training Environment?</a>.</td>
</tr>
<tr>
<td>Compute Nodes</td>
<td>Set the number of compute nodes. The default value is 1.</td>
</tr>
</tbody>
</table>

d. Configure the subscription function and set whether to save the parameter settings of the training job.

Figure 5-4 Configuring notifications for the training job
Table 5-23 Parameters related to the subscription function and parameter configuration saving

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Notification               | Select the resource pool status to be monitored from the event list, and SMN sends a notification message when the event occurs. This parameter is optional. You can choose whether to enable subscription based on actual requirements. If you enable subscription, set the following parameters as required:  
  - **Topic**: indicates the topic name. You can create a topic on the SMN console.  
  - **Event**: indicates the event to be subscribed to. The options are **OnJobRunning**, **OnJobSucceeded**, and **OnJobFailed**, indicating that the job is in progress, successful, and failed, respectively. |
| Saving Training Parameters | If you select this option, the parameter settings of the current job will be saved to facilitate subsequent job creation. Select **Save Training Parameters** and specify **Configuration Name** and **Description**. After a training job is created, you can switch to the **Job Parameters** tab page to view your saved job parameter settings. For details, see **Managing Job Parameters**. |

4. After setting the parameters, click **Next**.

5.3.3 Using Built-in Algorithms to Train Models

If you do not have the algorithm development capability, you can use the built-in algorithms of ModelArts. After simple parameter adjustment, you can create a training job and build a model.
Prerequisites

- Data has been prepared. Specifically, you have created an available dataset in ModelArts, or you have uploaded the dataset used for training to the OBS directory.
- At least one empty folder has been created on OBS for storing the training output.
- The account is not in arrears because resources are consumed when training jobs are running.
- The OBS directory you use and ModelArts are in the same region.

Precautions

In the dataset directory specified for a training job, the names of the files (such as the image file, audio file, and label file) containing data used for training contain 0 to 255 characters. If the names of certain files in the dataset directory contain over 255 characters, the training job will ignore these files and use data in the valid files for training. If the names of all files in the dataset directory contain over 255 characters, no data is available for the training job and the training job fails.

Creating a Training Job

1. Log in to the ModelArts management console. In the left navigation pane, choose Training Management > Training Jobs. By default, the system switches to the Training Jobs page.
2. In the upper left corner of the training job list, click Create to switch to the Create Training Job page.
3. Set related parameters and click Next.
   a. Set the basic information, including Billing Mode, Name, Version, and Description. Billing Mode supports only Pay-per-use. The Version information is automatically generated by the system and named in an ascending order of V001, V002, and so on. You cannot manually modify it.

   Specify Name and Description according to actual requirements.

   Figure 5-5 Setting basic information about the training job

   ![Figure 5-5 Setting basic information about the training job](image)

   b. Set job parameters, including the data source, algorithm source, and more. For details, see Table 5-24.
Figure 5-6 Built-in as the algorithm source

Table 5-24 Job parameter description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sub-Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-Click</td>
<td></td>
<td>If you have saved job parameter configurations in ModelArts, click <strong>One-Click Configuration</strong> and select an existing job parameter configuration as prompted to quickly complete parameter setting for the job.</td>
</tr>
<tr>
<td>Algorithm Source</td>
<td>Built-in</td>
<td>Select a built-in algorithm in ModelArts. For details, see <strong>Introduction to Built-in Algorithms</strong>.</td>
</tr>
<tr>
<td>Data Source</td>
<td>Dataset</td>
<td>Select an available dataset and its version from the ModelArts <strong>Data Management</strong> module.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Dataset</strong>: Select an existing dataset from the drop-down list. If no dataset is available in ModelArts, no result will be displayed in the drop-down list.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Version</strong>: Select a version according to the <strong>Dataset</strong> setting.</td>
</tr>
<tr>
<td>Data Source</td>
<td>Data path</td>
<td>Select the training data from your OBS bucket. On the right of the <strong>Data path</strong> text box, click <strong>Select</strong>. In the dialog box that is displayed, select an OBS folder for storing data. The dataset must meet the requirements of different types of built-in algorithms. For details, see <strong>Requirements on Datasets</strong>.</td>
</tr>
</tbody>
</table>
After you select a built-in algorithm, the running parameters that are set by default are displayed based on the selected algorithm. You can modify the parameters based on the actual requirements. For details about the running parameters of different algorithms, see *Algorithms and Their Running Parameters*. You can also use the default values to create a training job. If the training result is unsatisfactory, you can optimize the parameters.

**Training Output Path**

- Storage path of the training result

**NOTE**

To avoid errors, you are advised to select an empty directory for **Training Output Path**. Do not select the directory used for storing the dataset for **Training Output Path**.

**Job Log Path**

- Select a path for storing log files generated during job running.

c. Select resources for the training job.

**Figure 5-7** Selecting resources for the training job

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sub-Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running Parameter</td>
<td>-</td>
<td>After you select a built-in algorithm, the running parameters that are set by default are displayed based on the selected algorithm. You can modify the parameters based on the actual requirements. For details about the running parameters of different algorithms, see <em>Algorithms and Their Running Parameters</em>. You can also use the default values to create a training job. If the training result is unsatisfactory, you can optimize the parameters.</td>
</tr>
<tr>
<td>Training Output Path</td>
<td>-</td>
<td>Storage path of the training result <strong>NOTE</strong> To avoid errors, you are advised to select an empty directory for <strong>Training Output Path</strong>. Do not select the directory used for storing the dataset for <strong>Training Output Path</strong>.</td>
</tr>
<tr>
<td>Job Log Path</td>
<td>-</td>
<td>Select a path for storing log files generated during job running.</td>
</tr>
</tbody>
</table>

**Table 5-25** Resource parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Pool</td>
<td>Select resource pools for the job. For training jobs, <strong>Public resource pools</strong> and <strong>Dedicated resource pools</strong> are available. Instances in the public resource pool can be of the CPU or GPU type. Pricing standards for resource pools with different instance types are different. For details, see <strong>Product Pricing Details</strong>.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Type</td>
<td>If <strong>Resource Pool</strong> is set to <strong>Public resource pools</strong>, select a resource type. Available resource types are <strong>CPU</strong> and <strong>GPU</strong>. The GPU resource delivers better performance, and the CPU resource is more cost effective. If the selected algorithm has been defined to use the CPU or GPU, the resource type is automatically displayed on the page. Select the resource type as required. The data disk capacity varies depending on the resource type. For details, see <a href="#">What Are Sizes of the /cache Directories for GPU and CPU Resources in the Training Environment?</a>.</td>
</tr>
<tr>
<td>Specification</td>
<td>Select a resource flavor based on the resource type.</td>
</tr>
<tr>
<td>Compute Nodes</td>
<td>Set the number of compute nodes. If you set <strong>Compute Nodes</strong> to 1, the standalone computing mode is used. If you set <strong>Compute Nodes</strong> to a value greater than 1, the distributed computing mode is used. Currently, only the <code>modelarts.bm.gpu.8v100</code> flavor supports distributed training.</td>
</tr>
</tbody>
</table>

d. Configure **Notification** and select whether to save the parameters of the training job.

**Figure 5-8** Configuring notifications for the training job

![Notification Configuration](image-url)
Table 5-26 Parameters related to the subscription function and parameter configuration saving

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Notification       | Select the resource pool status to be monitored from the event list, and SMN sends a notification message when the event occurs. This parameter is optional. You can choose whether to enable subscription based on actual requirements. If you enable subscription, set the following parameters as required:  
  - **Topic**: indicates the topic name. You can create a topic on the SMN console.  
  - **Event**: indicates the event to be subscribed to. The options are OnJobRunning, OnJobSucceeded, and OnJobFailed, indicating that training is in progress, successful, and failed, respectively. |
| Saving Training Parameters | If you select this option, the parameter settings of the current training will be saved to facilitate subsequent job creation. Select **Save Training Parameters** and specify **Configuration Name** and **Description**. After a training job is created, you can switch to the **Job Parameters** tab page to view your saved job parameter settings. For details, see **Managing Job Parameters**. |

4. After setting the parameters, click **Next**.

5. Confirm that the information is correct on the **Confirm** page that is displayed and click **Submit**. Generally, training jobs run for a period of time, which may be several minutes or tens of minutes depending on the amount of your selected data and resources.

**NOTE**

After a training job is created, it is started immediately. During the running, you will be charged based on your selected resources.

You can switch to the training job list to view the basic information about training jobs. In the training job list, **Status** of the newly created training job is **Initializing**. If the status changes to **Successful**, the training job ends and the model generated is stored in the location specified by **Training Output Path**. If the status of a training job changes to **Running failed**, click the name of the training job and view the job logs. Troubleshoot the fault based on the logs.

### 5.3.4 Using Frequently-used Frameworks to Train Models

If you use frequently-used frameworks, such as TensorFlow and MXNet, to develop algorithms locally, you can select **Frequently-used** to create training jobs and build models.
Prerequisites

- Data has been prepared. Specifically, you have created an available dataset in ModelArts, or you have uploaded the dataset used for training to the OBS directory.
- If you select Frequently-used for Algorithm Source, prepare the training script and upload it to the OBS directory.
- At least one empty folder has been created on OBS for storing the training output.
- The account is not in arrears because resources are consumed when training jobs are running.
- The OBS directory you use and ModelArts are in the same region.

Precautions

- In the dataset directory specified for a training job, the names of the files (such as the image file, audio file, and label file) containing data used for training contain 0 to 255 characters. If the names of certain files in the dataset directory contain over 255 characters, the training job will ignore these files and use data in the valid files for training. If the names of all files in the dataset directory contain over 255 characters, no data is available for the training job and the training job fails.
- In the training script, the Data Source and Training Output Path parameters must be set to the OBS path. You are advised to use the MoXing interface to perform read and write operations in the path.

Frequently-used AI Frameworks for Training Management

ModelArts supports the following AI engines and versions.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>CPU/GPU</td>
<td>x86_64</td>
<td>Ubuntu16.04</td>
<td>TF-1.8.0-python3.6</td>
<td>cuda9.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TF-1.8.0-python2.7</td>
<td>cuda9.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TF-1.13.1-python3.6</td>
<td>cuda10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TF-1.13.1-python2.7</td>
<td>cuda10.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TF-2.1.0-python3.6</td>
<td>cuda10.1</td>
</tr>
<tr>
<td>MXNet</td>
<td>CPU/GPU</td>
<td>x86_64</td>
<td>Ubuntu16.04</td>
<td>MXNet-1.2.1-python3.6</td>
<td>cuda9.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MXNet-1.2.1-python2.7</td>
<td>cuda9.0</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
<td>---------------------</td>
<td>----------------</td>
<td>----------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Caffe</td>
<td>CPU/GPU</td>
<td>x86_64</td>
<td>Ubuntu16.04</td>
<td>Caffe-1.0.0-python2.7</td>
<td>cuda8.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spark_MLLib</td>
<td>CPU</td>
<td>x86_64</td>
<td>Ubuntu16.04</td>
<td>Spark-2.3.2-python2.7</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ray</td>
<td>CPU/GPU</td>
<td>x86_64</td>
<td>Ubuntu16.04</td>
<td>RAY-0.7.4-python3.6</td>
<td>cuda10.0</td>
</tr>
<tr>
<td>XGBoost-Sklearn</td>
<td>CPU</td>
<td>x86_64</td>
<td>Ubuntu16.04</td>
<td>Scikit_Learn-0.18.1-python2.7</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyTorch</td>
<td>CPU/GPU</td>
<td>x86_64</td>
<td>Ubuntu16.04</td>
<td>PyTorch-1.0.0-python2.7</td>
<td>cuda9.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PyTorch-1.0.0-python3.6</td>
<td>cuda9.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ascend-Powered-Engine</td>
<td>Ascend 910</td>
<td>aarch64</td>
<td>Euler2.8</td>
<td>Mindspore-1.0-c75-python3.7-euleros2.8-aarch64</td>
<td>C75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TF-1.15-python3.7-aarch64</td>
<td>C75</td>
</tr>
<tr>
<td>MindSpore-GPU</td>
<td>CPU/GPU</td>
<td>x86_64</td>
<td>Ubuntu18.04</td>
<td>MindSpore-1.0.1-python3.7</td>
<td>cuda10.1</td>
</tr>
</tbody>
</table>

**NOTE**

- MoXing is a distributed training acceleration framework developed by the ModelArts team. It is built on the open source deep learning engines TensorFlow, MXNet, PyTorch, and Keras. For details about how to use MoXing, see **MoXing Framework**. If you use MoXing to compile a training script, select the corresponding AI engine and version based on your selected API when creating a training job.
- If the version supported by your training script is different the AI engine and version used in the training job, the training may fail.
Creating a Training Job

1. Log in to the ModelArts management console. In the left navigation pane, choose Training Management > Training Jobs. By default, the system switches to the Training Jobs page.

2. In the upper left corner of the training job list, click Create to switch to the Create Training Job page.

3. Set related parameters and click Next.
   a. Set the basic information, including Billing Mode, Name, Version, and Description. Billing Mode supports only Pay-per-use. The Version information is automatically generated by the system and named in an ascending order of V001, V002, and so on. You cannot manually modify it.

   Specify Name and Description according to actual requirements.

   Figure 5-9 Setting basic information about the training job

   ![Set basic information about the training job](image)

   b. Set job parameters, including the data source, algorithm source, and more. For details, see Table 5-28.

   Figure 5-10 Frequently-used as the algorithm source

   ![Frequently-used as the algorithm source](image)
### Table 5-28 Job parameter description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sub-Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-Click Configuration</td>
<td>-</td>
<td>If you have saved job parameter configurations in ModelArts, click One-Click Configuration and select an existing job parameter configuration as prompted to quickly complete parameter setting for the job.</td>
</tr>
<tr>
<td>Algorithm Source</td>
<td>Frequently-used</td>
<td>Select an AI engine and its version and specify Code Directory and Boot File. The framework selected for the AI engine must be the same as the one you select when compiling training code. For example, if TensorFlow is used in your training code, select TensorFlow when you create a training job. For details about the supported AI engines and versions, see Frequently-used AI Frameworks for Training Management. If your model requires Python dependencies, place the dependency package and its configuration file in the code directory based on the requirements defined in ModelArts. For details, see How Do I Create a Training Job When a Dependency Package is Referenced in a Model.</td>
</tr>
</tbody>
</table>
| Data Source             | Dataset       | Select an available dataset and its version from the ModelArts Data Management module.  
|                         |               | - Dataset: Select an existing dataset from the drop-down list. If no dataset is available in ModelArts, no result will be displayed in the drop-down list.  
|                         |               | - Version: Select a version according to the Dataset setting. |
|                         | Data path     | Select the training data from your OBS bucket. On the right of the Data path text box, click Select. In the dialog box that is displayed, select an OBS folder for storing data. |
|                         | Training Output Path | Select a path for storing the training result.  
|                         |               | **NOTE**  
|                         |               | To avoid errors, you are advised to select an empty directory for Training Output Path. Do not select the directory used for storing the dataset for Training Output Path. |
### Table 5-29 Resource parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Pool</td>
<td>Select resource pools for the job. Instances in the public resource pool can be of the CPU or GPU type. Pricing standards for resource pools with different instance types are different. For details, see <a href="#">Product Pricing Details</a>.</td>
</tr>
<tr>
<td>Type</td>
<td>If Resource Pool is set to Public resource pools, select a resource type. Available resource types are CPU and GPU. The GPU resource delivers better performance, and the CPU resource is more cost effective. If the selected algorithm has been defined to use the CPU or GPU, the resource type is automatically displayed on the page. Select the resource type as required. The data disk capacity varies depending on the resource type. For details, see <a href="#">What Are Sizes of the /cache Directories for GPU and CPU Resources in the Training Environment?</a>.</td>
</tr>
</tbody>
</table>

**NOTE**

If GPU resources are used in training code, you must select a GPU cluster when selecting a resource pool. Otherwise, the training job may fail.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifications</td>
<td>Select a resource flavor based on the resource type.</td>
</tr>
<tr>
<td>Compute Nodes</td>
<td>Set the number of compute nodes. If you set <strong>Compute Nodes</strong> to 1, the standalone computing mode is used. If you set <strong>Compute Nodes</strong> to a value greater than 1, the distributed computing mode is used. Select a computing mode based on the actual requirements. When <strong>Frequently-used</strong> of <strong>Algorithm Source</strong> is set to <strong>Caffe</strong>, only standalone training is supported, that is, <strong>Compute Nodes</strong> must be set to 1. For other options of <strong>Frequently-used</strong>, you can select the standalone or distributed mode based on service requirements.</td>
</tr>
</tbody>
</table>

**Figure 5-12** Configuring notifications for the training job

**Table 5-30** Parameters related to notification and parameter configuration saving

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Notification    | Select the resource pool status to be monitored from the event list, and SMN sends a notification message when the event occurs. This parameter is optional. You can choose whether to enable subscription based on actual requirements. If you enable subscription, set the following parameters as required: 
  - **Topic**: indicates the topic name. You can create a topic on the SMN console. 
  - **Event**: indicates the event to be subscribed to. The options are **OnJobRunning**, **OnJobSucceeded**, and **OnJobFailed**, indicating that training is in progress, successful, and failed, respectively. |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saving Training Parameters</td>
<td>If you select this option, the parameter settings of the current job will be saved to facilitate subsequent job creation.</td>
</tr>
<tr>
<td></td>
<td>Select <strong>Save Training Parameters</strong> and specify <strong>Configuration Name</strong> and <strong>Description</strong>. After a training job is created, you can switch to the <strong>Job Parameters</strong> tab page to view your saved job parameter configuration. For details, see <strong>Managing Job Parameters</strong>.</td>
</tr>
</tbody>
</table>

4. After setting the parameters, click **Next**.

4. Confirm that the information is correct on the **Confirm** page that is displayed and click **Submit**. Generally, training jobs run for a period of time, which may be several minutes or tens of minutes depending on the amount of your selected data and resources.

**NOTE**

After a training job is created, it is started immediately. During the running, you will be charged based on your selected resources.

You can switch to the training job list to view the basic information about training jobs. In the training job list, **Status** of the newly created training job is **Initializing**. If the status changes to **Successful**, the training job ends and the model generated is stored in the location specified by **Training Output Path**. If the status of a training job changes to **Running failed**, click the name of the training job and view the job logs. Troubleshoot the fault based on the logs.

5.3.5 Using Custom Images to Train Models

If the framework used for algorithm development is not a frequently-used framework, you can build an algorithm into a custom image and use the custom image to create a training job.

**Prerequisites**

- Data has been prepared. Specifically, you have created an available dataset in ModelArts, or you have uploaded the dataset used for training to the OBS directory.
- If the algorithm source is **Custom**, create an image and upload the image to SWR. For details, see **Creating and Uploading a Custom Image**.
- The training script has been uploaded to the OBS directory.
- At least one empty folder has been created on OBS for storing the training output.
- The account is not in arrears because resources are consumed when training jobs are running.
- The OBS directory you use and ModelArts are in the same region.
Precautions

- In the dataset directory specified for a training job, the names of the files (such as the image file, audio file, and label file) containing data used for training contain 0 to 255 characters. If the names of certain files in the dataset directory contain over 255 characters, the training job will ignore these files and use data in the valid files for training. If the names of all files in the dataset directory contain over 255 characters, no data is available for the training job and the training job fails.

- In the training script, the Data Source and Training Output Path parameters must be set to the OBS path. You are advised to use the MoXing interface to perform read and write operations in the path.

Creating a Training Job

1. Log in to the ModelArts management console. In the left navigation pane, choose Training Management > Training Jobs. By default, the system switches to the Training Jobs page.

2. In the upper left corner of the training job list, click Create to switch to the Create Training Job page.

3. Set related parameters and click Next.

   a. Set the basic information, including Billing Mode, Name, Version, and Description. Billing Mode supports only Pay-per-use. The Version information is automatically generated by the system and named in an ascending order of V001, V002, and so on. You cannot manually modify it.

   Specify Name and Description according to actual requirements.

   Figure 5-13 Setting basic information about the training job

   

   ![Figure 5-13](image)

   b. Set job parameters, including the data source, algorithm source, and more. For details, see Table 5-31.
Figure 5-14 Custom as the algorithm source

Table 5-31 Job parameter description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sub-Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-Click Configuration</td>
<td>-</td>
<td>If you have saved job parameter configurations in ModelArts, click One-Click Configuration and select an existing job parameter configuration as prompted to quickly complete parameter setting for the job.</td>
</tr>
<tr>
<td>Algorithm Source</td>
<td>Custom</td>
<td>For details about custom image specifications, see Specifications for Custom Images Used for Training Jobs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Image Path: SWR URL generated after the image is uploaded to SWR. For details, see 3.b.i.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Code Directory: OBS path for storing the training code file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Boot Command: boot command after the image is started. For details, see 3.b.ii.</td>
</tr>
<tr>
<td>Data Source</td>
<td>Dataset</td>
<td>Select an available dataset and its version from the ModelArts Data Management module.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Dataset: Select an existing dataset from the drop-down list. If no dataset is available in ModelArts, no result will be displayed in the drop-down list.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Version: Select a version according to the Dataset setting.</td>
</tr>
</tbody>
</table>
Pay special attention to the settings of the following job parameters:

i. **Image Path**
   SWR URL generated after the image is uploaded to SWR.
   **Figure 5-15 SWR image address**

ii. **Boot Command**
   Boot command after the image is started. The basic format is as follows:
   ```bash
   /home/work/run_train.sh {UserCommand}
   /home/work/run_train.sh [python/bash/..] {file_location} {file_parameter}
   ```
   **run_train.sh** is the training boot script. After this script is executed, ModelArts recursively downloads all content in the code directory to the local path of the container. The local path is in the format of /
home/work/user-job-dir/${Name of the last level in the code directory}.

For example, if the OBS path of the training code file is obs://obs-bucket/new/train.py and the code directory is obs://obs-bucket/new/, the local path of the container is /home/work/user-job-dir/new/. The local training code path of the container is /home/work/user-job-dir/new/train.py. Then, you can set the boot command to the following: bash /home/work/run_train.sh python /home/work/user-job-dir/new/train.py {python_file_parameter}

**NOTE**

If you create a training job using a custom image, ModelArts allows you to customize the boot command. The following are two basic formats for the boot command:

```
bash /home/work/run_train.sh {UserCommand}
bash /home/work/run_train.sh [python/bash/..] {file_location} {file_parameter}
```

`run_train.sh` is the training boot script. When creating a custom image, you can implement the training boot script or place the training code in the custom image environment in advance to customize the boot command (in the basic formats or any other formats).

### iii. Environment Variable

After the container is started, besides the environment variables added by configuring Environment Variable during training job creation, Table 5-32 lists other environment variables to be loaded. You can determine whether to use these environment variables in your own Python training script, or run the `{python_file_parameter}` command to pass the required parameters.

**Table 5-32 Environment variables**

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLS_TASK_INDEX</td>
<td>Container index, starting from 0.</td>
</tr>
<tr>
<td>DLS_TASK_NUMBER</td>
<td>Number of containers, corresponding to Compute Nodes</td>
</tr>
<tr>
<td>DLS_APP_URL</td>
<td>Code directory, corresponding to Code Dir with the protocol name added. For example, you can use SDLS_APP_URL/*.py to read files in OBS.</td>
</tr>
<tr>
<td>DLS_DATA_URL</td>
<td>Dataset path, corresponding to Data Source with the protocol name added</td>
</tr>
<tr>
<td>DLS_TRAIN_URL</td>
<td>Training output path, corresponding to Training Output Path with the protocol name added</td>
</tr>
</tbody>
</table>
### Environment Variable

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATCH_{jobName}.0_HOSTS (standalone)</td>
<td>For standalone training, that is, when the number of compute nodes is 1, the environment variable is BATCH_{jobName}.0_HOSTS. The format of the HOSTS environment variable is hostname:port. A container can view the HOSTS of all containers in the same job, such as BATCH_CUSTOM0_HOSTS and BATCH_CUSTOM1_HOSTS, varying according to the indexes. If the resource pool is a dedicated resource pool with the 8GPU specifications, the network type of the container is a host network, and the host IB network can be used to accelerate communications. If other resource pools are used, the network is a container network. <strong>NOTE</strong> When the host IB network is used for communication acceleration, the ip_mapper.py tool is required to obtain the IP address of the ib0 NIC for using the IPoIB feature.</td>
</tr>
</tbody>
</table>

#### c. Select resources for the training job.

**Figure 5-16** Selecting resources for the training job

![Resource selection interface](image)

**Table 5-33** Resource parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Pool</td>
<td>Select resource pools for the job. Instances in the public resource pool can be of the CPU or GPU type. Pricing standards for resource pools with different instance types are different. For details, see <strong>Product Pricing Details</strong>.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Type</td>
<td>If Resource Pool is set to Public resource pools, select a resource type. Available resource types are CPU and GPU. The GPU resource delivers better performance, and the CPU resource is more cost effective. If the selected algorithm has been defined to use the CPU or GPU, the resource type is automatically displayed on the page. Select the resource type as required. The data disk capacity varies depending on the resource type. For details, see What Are Sizes of the /cache Directories for GPU and CPU Resources in the Training Environment?</td>
</tr>
<tr>
<td>Specifications</td>
<td>Select a resource flavor based on the resource type.</td>
</tr>
<tr>
<td>Compute Nodes</td>
<td>Set the number of compute nodes. If you set Compute Nodes to 1, the standalone computing mode is used. If you set Compute Nodes to a value greater than 1, the distributed computing mode is used. Select a computing mode based on the actual requirements.</td>
</tr>
</tbody>
</table>

d. Configure Notification and select whether to save the parameters of the training job.

**Figure 5-17 Configuring notifications for the training job**
### Table 5-34 Parameters related to notification and parameter configuration saving

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notification</strong></td>
<td>Select the resource pool status to be monitored from the event list, and SMN sends a notification message when the event occurs. This parameter is optional. You can choose whether to enable subscription based on actual requirements. If you enable subscription, set the following parameters as required:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Topic</strong>: indicates the topic name. You can create a topic on the SMN console.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Event</strong>: indicates the event to be subscribed to. The options are <strong>OnJobRunning</strong>, <strong>OnJobSucceeded</strong>, and <strong>OnJobFailed</strong>, indicating that training is in progress, successful, and failed, respectively.</td>
</tr>
<tr>
<td><strong>Saving Training Parameters</strong></td>
<td>If you select this option, the parameter settings of the current job will be saved to facilitate subsequent job creation. Select <strong>Save Training Parameters</strong> and specify <strong>Configuration Name</strong> and <strong>Description</strong>. After a training job is created, you can switch to the <strong>Job Parameters</strong> tab page to view your saved job parameter settings. For details, see <strong>Managing Job Parameters</strong>.</td>
</tr>
</tbody>
</table>

e. After setting the parameters, click **Next**.

4. Confirm that the information is correct on the **Confirm** page that is displayed and click **Submit**. Generally, training jobs run for a period of time, which may be several minutes or tens of minutes depending on the amount of your selected data and resources.

After a custom image job is created, the system authorizes ModelArts to obtain and run the image by default. When you run a custom image job for the first time, ModelArts checks the custom image. For details about the check, see **Specifications for Custom Images Used for Training Jobs**. You can view the cause of the check failure in the log and modify the custom image based on the log.

**Figure 5-18** Failed to check the image
After the image is checked, the background starts the custom image container to run the custom image training job. You can switch to the training job list to view the basic information about training jobs. In the training job list, **Status** of the newly created training job is **Initializing**. If the status changes to **Successful**, the training job ends and the model generated is stored in the location specified by **Training Output Path**. If the status of a training job changes to **Running failed**, click the name of the training job and view the job logs. Troubleshoot the fault based on the logs.

![Figure 5-19 Run log](image)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Logs</th>
<th>Resource Usage</th>
<th>Evaluation Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log File</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2021-10-08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2021-10-08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2021-10-08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2021-10-08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2021-10-08</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE**
- After a training job is created, it is started immediately. During the running, you will be charged based on your selected resources.
- After an image is reviewed, the image does not need to be reviewed again when being used to create training jobs again.

### 5.4 Stopping or Deleting a Job

#### Stopping a Training Job

In the training job list, click **Stop** in the **Operation** column for a training job in the **Running** state to stop a running training job.

After the training job is stopped, its billing stops on ModelArts. If you have selected **Save Training Parameters** for a stopped training job, the job’s parameter settings will be saved to the **Job Parameter Mgmt** page.

You cannot stop a training job that has stopped running, for example the job in the **Successful** or **Running failed** state. Only training jobs in the **Running** state can be stopped.

#### Deleting a Training Job

If an existing training job is no longer used, you can delete it.

For a training job in the **Running**, **Successful**, **Running failed**, **Canceled**, or **Deploying** state, click **Delete** in the **Operation** column to delete it.

If you have selected **Save Training Parameters** for a deleted training job, the job’s parameter settings will be saved to the **Job Parameter Mgmt** page.
5.5 Managing Training Job Versions

During model building, you may need to frequently tune the data, training parameters, or the model based on the training results to obtain a satisfactory model. ModelArts allows you to manage training job versions to effectively train your model after the tuning. Specifically, ModelArts generates a version each time when a training is performed. You can quickly get the difference between different versions.

Viewing Training Job Versions

1. Log in to the ModelArts management console. In the left navigation pane, choose Training Management > Training Jobs. By default, the system switches to the Training Jobs page.
2. In the training job list, click the name of a training job.
   By default, the basic information about the latest version is displayed. If there are multiple available versions, click Select Version in the upper left corner to view a certain version. Click the downward arrow to the left of the version to display job details. For details about the training job, see Training Job Details.

![Figure 5-20 Viewing training job versions](image)

Comparing Versions of a Training Job

On the Version Manager page, click View Comparison Result to view the comparison of all or selected versions of the current training job. The comparison result involves the following information: Running Parameter, F1 Score, Recall, Precision, and Accuracy.

护卫

The F1 Score, Recall, Precision, and Accuracy parameters of a training job are displayed only when the job is created using a built-in algorithm. For training jobs created using frequently-used frameworks or custom images, define the output of these parameters in your training script code. These parameters cannot be viewed on the GUI.
Shortcut Operations Based on Training Job Versions

On the Version Manager page, ModelArts provides certain shortcut operation buttons for you to quickly enter the subsequent steps after model training is complete.

Table 5-35 Shortcut operation button description

<table>
<thead>
<tr>
<th>Shortcut Operation Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating Visualization Job</td>
<td>Creates a visualization job (TensorBoard) for the current training version. For details about how to create a TensorBoard job, see Managing Visualization Jobs. <strong>NOTE</strong> Currently, TensorBoard supports only the TensorFlow and MXNet engines. Therefore, you can create the TensorBoard jobs only when the AI engine is TensorFlow or MXNet.</td>
</tr>
<tr>
<td>Create Model</td>
<td>Creates a model for the current training version. For details about how to create a model, see Importing a Model. You can only create models for training jobs in the Running status.</td>
</tr>
<tr>
<td>Modify</td>
<td>If the training result of the current version does not meet service requirements or the training job fails, click Modify to switch to the page where you can modify the job parameter settings. For details about the parameters of the training job, see Creating a Training Job. After modifying the job parameter settings as required, click OK to start the training job of a new version.</td>
</tr>
</tbody>
</table>
5.6 Viewing Job Details

After a training job finishes, you can manage the training job versions and check whether the training result of the job is satisfactory by viewing the job details and Viewing the Evaluation Result.

Training Job Details

In the left navigation pane of the ModelArts management console, choose Training Management > Training Jobs to switch to the Training Jobs page. In the training job list, click a job name to view the job details.

Table 5-36 lists parameters of the training job of each version.
Figure 5-23 Training job details

![Training job details](image)

Table 5-36 Training job details

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Version of a training job, which is automatically defined by the system, for example, <strong>V0001</strong> and <strong>V0002</strong>.</td>
</tr>
<tr>
<td>Status</td>
<td>Status of a training job,</td>
</tr>
<tr>
<td>Duration</td>
<td>Running duration of a training job</td>
</tr>
<tr>
<td>Configurations</td>
<td>Details about the parameters of the current training job version</td>
</tr>
<tr>
<td>Logs</td>
<td>Logs of the current training job version. If you set <strong>Log Output Path</strong> when creating a training job, you can click the download button on the <strong>Logs</strong> tab page to download the logs stored in the OBS bucket to the local host.</td>
</tr>
<tr>
<td>Resource Usages</td>
<td>Usage of resources of the current training version, including the CPU, GPU, and memory.</td>
</tr>
</tbody>
</table>

**Viewing the Evaluation Result**

When you use a built-in algorithm published by ModelArts to create a training job, you can view the evaluation result of the training job. If the evaluation code is added to the training script based on the ModelArts specifications, you can view the evaluation result on the job details page after the training job is complete. For details about how to add the evaluation code, see **Adding the Evaluation Code**.

**NOTE**

Currently, you can view evaluation results of models using algorithms such as **Image Classification-ResNet_v1_50**, **Object Detection-FasterRCNN_ResNet50**, and **Object Detection-EfficientDet** built-in algorithms.
After a training job is executed and its status is **Successful**, you can view the evaluation details on the **Evaluation Result** tab page of the job details page. See **Figure 5-24**.

The evaluation results include the evaluation overview, precision evaluation, sensitivity analysis, and common model indicators. The system automatically provides optimization suggestions based on your model metrics. Read the suggestions and guidance on the page carefully to further optimize your model.

![Figure 5-24 Viewing the evaluation result](image)

### 5.7 Managing Job Parameters

You can store the parameter settings in ModelArts during job creation so that you can use the stored settings to create follow-up training jobs, which makes job creation more efficient.

During the operations of creating, editing, and viewing training jobs, the saved job parameter settings are displayed on the **Job Parameter Mgmt** page.

#### Using a Job Parameter Configuration

- **Method 1**: Using a job parameter configuration on the **Job Parameter Mgmt** page
  
  Log in to the ModelArts management console. In the left navigation pane, choose **Training Management > Training Jobs**. On the displayed page, click the **Job Parameter Mgmt** tab. In the job parameter list, click **Creating Training Job** for a job parameter configuration to create a training job based on the job parameter configuration.

- **Method 2**: Using a job parameter configuration on the **Creating Training Job** page
  
  On the **Creating Training Job** page, click **One-Click Configuration**. In the displayed dialog box, select the required job parameter configuration to quickly create an available training job.

#### Editing a Job Parameter Configuration

1. Log in to the ModelArts management console. In the left navigation pane, choose **Training Management > Training Jobs**. On the displayed page, click the **Job Parameter Mgmt** tab.
2. In the job parameter configuration list, click **Edit** in the **Operation** column in a row.

3. On the displayed page, modify related parameters by referring to **Creating a Training Job** and click **OK** to save the job parameter settings.
   In the existing job parameter settings, the job name cannot be changed.

**Deleting a Training Job Parameter Configuration**

1. Log in to the ModelArts management console. In the left navigation pane, choose **Training Management > Training Jobs**. On the displayed page, click the **Job Parameter Mgmt** tab.

2. In the job parameter list, click **Delete** in the **Operation** column in a row.

3. In the displayed dialog box, click **OK**.

   **NOTE**

   Deleted job parameter configurations cannot be recovered. Therefore, exercise caution when performing this operation.

### 5.8 Adding the Evaluation Code

After a training job is executed, ModelArts automatically evaluates your model and provides optimization diagnosis and suggestions. For details, see **Viewing the Evaluation Result**.

- When you use a built-in algorithm to create a training job, you can view the evaluation result without any configurations.
- For a training job created by compiling a training script or using a custom image, you need to add the evaluation code to the training code so that you can view the evaluation result and diagnosis suggestions after the training job is complete.

   **NOTE**

   1. Only validation sets of the image type are supported.
   2. Currently, you can add the evaluation code only when the training scripts of the following frequently-used frameworks are used:

   - TF-1.13.1-python3.6
   - TF-2.1.0-python3.6
   - PyTorch-1.4.0-python3.6

This section describes how to use the evaluation code in a training job. To adapt and modify the training code, three steps are involved, adding the output path, **copying the dataset to the local host**, and **mapping the dataset path to OBS**.

#### Adding the Output Path

The code for adding the output path is simple. That is, add a path for storing the evaluation result file to the code, which is called **train_url**, that is, the training output path on the console. Add **train_url** to the analysis function and use **save_path** to obtain **train_url**. The sample code is as follows:

```python
FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string('model_url', '', 'path to saved model')
```
tf.app.flags.DEFINE_string('data_url', '', 'path to output files')
tf.app.flags.DEFINE_string('train_url', '', 'path to output files')
tf.app.flags.DEFINE_string('adv_param_json', '{"attack_method":"FGSM","eps":40},
  "params for adversarial attacks")
FLAGS(sys.argv, known_only=True)

# analyse
res = analyse(
    task_type=task_type,
    pred_list=pred_list,
    label_list=label_list,
    name_list=file_name_list,
    label_map_dict=label_dict,
    save_path=FLAGS.train_url)

### Copying the Dataset to the Local Host

Copying a dataset to the local host is to prevent the OBS connection from being interrupted due to long-time access. Therefore, you are advised to copy the dataset to the local host before performing operations.

Datasets can be copied in two modes. You are advised to use the OBS path to copy datasets.

- **OBS path (recommended)**
  
  Call the copy_parallel API of MoXing to copy the corresponding OBS path.

- **Dataset in ModelArts data management (manifest file format)**
  
  Call the copy_manifest API of MoXing to copy the file to the local host and obtain the path of the new manifest file. Then, use SDK to parse the new manifest file.

```python
if data_path.startswith('obs://'):
    if '.manifest' in data_path:
        new_manifest_path, _ = mox.file.copy_manifest(data_path, '/cache/data/)
        data_path = new_manifest_path
    else:
        mox.file.copy_parallel(data_path, '/cache/data/)
        data_path = '/cache/data/'
print('------------- download dataset success ------------')
```

### Mapping the Dataset Path to OBS

The actual path of the image file, that is, the OBS path, needs to be entered in the JSON body. Therefore, after analysis and evaluation are performed on the local host, the original local dataset path needs to be mapped to the OBS path, and the new list needs to be sent to the analysis interface.

If the OBS path is used as the input of `data_url`, you only need to replace the character string of the local path.

```python
if FLAGS.data_url.startswith('obs://'):
    for idx, item in enumerate(file_name_list):
        file_name_list[idx] = item.replace(data_path, FLAGS.data_url)
```

If the manifest file is used, the original manifest file needs to be parsed again to obtain the list and then the list is sent to the analysis interface.

```python
if or FLAGS.data_url.startswith('obs://'):
    if 'manifest' in FLAGS.data_url:
```
The sample code for image classification that adapts to a training job is as follows:

```python
import json
import logging
import os
import sys
import tensorflow as tf
import moxing as mox
from PIL import Image
import numpy as np
from deep_moxing.framework.manifest_api.manifest_api import get_sample_list
from deep_moxing.model_analysis.api import analyse, tmp_save
from deep_moxing.model_analysis.common.constant import TMP_FILE_NAME
logging.basicConfig(level=logging.DEBUG)
FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string('model_url', '', 'path to saved model')
tf.app.flags.DEFINE_string('data_url', '', 'path to output files')
FLAGS(sys.argv, known_only=True)
def _preprocess(data_path):
    img = Image.open(data_path)
    img = img.convert('RGB')
    img = np.asarray(img, dtype=np.float32)
    img = img[0, :, :, :
    return img
def softmax(x):
    x = np.array(x)
    orig_shape = x.shape
    if len(x.shape) > 1:
        # Matrix
        x = np.apply_along_axis(lambda x: np.exp(x - np.max(x)), 1, x)
        denominator = np.apply_along_axis(lambda x: 1.0 / np.sum(x), 1, x)
        if len(denominator.shape) == 1:
            denominator = denominator.reshape((denominator.shape[0], 1))
        x = x * denominator
    else:
        # Vector
        x_max = np.max(x)
        x = x - x_max
        numerator = np.exp(x)
        denominator = 1.0 / np.sum(numerator)
        x = numerator.dot(denominator)
    assert x.shape == orig_shape
    return x
def get_dataset(data_path, label_map_dict):
    label_list = []
    img_name_list = []
    if 'manifest' in data_path:
```
manifest, _ = get_sample_list(
    manifest_path=data_path, task_type='image_classification')
for item in manifest:
    if len(item[1]) != 0:
        label_list.append(label_map_dict.get(item[1][0]))
        img_name_list.append(item[0])
    else:
        continue
else:
    label_name_list = os.listdir(data_path)
    label_dict = {}
    for idx, item in enumerate(label_name_list):
        label_dict[str(idx)] = item
        sub_img_list = os.listdir(os.path.join(data_path, item))
        img_name_list += [os.path.join(data_path, item, img_name) for img_name in sub_img_list]
    label_list += [label_map_dict.get(item)] * len(sub_img_list)
return img_name_list, label_list

def deal_ckpt_and_data_with_obs():
    pb_dir = FLAGS.model_url
    data_path = FLAGS.data_url
    if pb_dir.startswith('obs://'):
        mox.file.copy_parallel(pb_dir, '/cache/ckpt/')
        pb_dir = '/cache/ckpt'
        print('------------- download success ------------
    if data_path.startswith('obs://'):
        if '.manifest' in data_path:
            new_manifest_path, _ = mox.file.copy_manifest(data_path, '/cache/data/')
            data_path = new_manifest_path
        else:
            mox.file.copy_parallel(data_path, '/cache/data/)
            data_path = '/cache/data/'
            print('------------- download dataset success ------------
return pb_dir, data_path

def evalution():
    pb_dir, data_path = deal_ckpt_and_data_with_obs()
    index_file = os.path.join(pb_dir, 'index')
    try:
        label_file = h5py.File(index_file, 'r')
        label_array = label_file['labels_list'][:].tolist()
        label_array = [item.decode('utf-8') for item in label_array]
    except Exception as e:
        logging.warning(e)
        logging.warning('index file is not a h5 file, try json.'
        with open(index_file, 'r') as load_f:
            label_file = json.load(load_f)
        label_array = label_file['labels_list'][:]
    label_map_dict = {}
    label_dict = {}
    for idx, item in enumerate(label_array):
        label_map_dict[item] = idx
        label_dict[idx] = item
    print(label_map_dict)
    print(label_dict)

data_file_list, label_list = get_dataset(data_path, label_map_dict)
assert len(label_list) > 0, 'missing valid data'
assert None not in label_list, 'dataset and model not match'
pred_list = []
file_name_list = []
img_list = []
for img_path in data_file_list:
    img = _preprocess(img_path)
    img_list.append(img)
    file_name_list.append(img_path)

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.visible_device_list = '0'
with tf.Session(graph=tf.Graph(), config=config) as sess:
    meta_graph_def = tf.saved_model.loader.load(
        sess, [tf.saved_model.tag_constants.SERVING], pb_dir)
    signature = meta_graph_def.signature_def
    signature_key = 'predict_object'
    input_key = 'images'
    output_key = 'logits'
    x_tensor_name = signature[signature_key].inputs[input_key].name
    y_tensor_name = signature[signature_key].outputs[output_key].name
    x = sess.graph.get_tensor_by_name(x_tensor_name)
    y = sess.graph.get_tensor_by_name(y_tensor_name)
    for img in img_list:
        pred_output = sess.run([y], {x: img})
        pred_output = softmax(pred_output[0])
        pred_list.append(pred_output[0].tolist())
    label_dict = json.dumps(label_dict)
    task_type = 'image_classification'
    if FLAGS.data_url.startswith('obs://'):
        if 'manifest' in FLAGS.data_url:
            file_name_list = []
            manifest, _ = get_sample_list(
                manifest_path=FLAGS.data_url,
                task_type='image_classification')
            for item in manifest:
                if len(item[1]) != 0:
                    file_name_list.append(item[0])
            for idx, item in enumerate(file_name_list):
                file_name_list[idx] = item.replace(data_path, FLAGS.data_url)
        # analyse
        res = analyse(
            task_type=task_type,
            pred_list=pred_list,
            label_list=label_list,
            name_list=file_name_list,
            label_map_dict=label_dict,
            save_path=FLAGS.train_url)
    if __name__ == "__main__":
evaluation()
Prerequisites

To ensure that the summary file is generated in the training result, you need to add the related code to the training script.

- Using the TensorFlow engine:
  When using the TensorFlow-based MoXing, in `mox.run`, set `save_summary_steps>0` and `summary_verbosity≥1`.
  If you want to display other metrics, add tensors to `log_info` in the return value `mox.ModelSpec` of `model_fn`. Only the rank-0 tensors (scalars) are supported. The added tensors are written into the summary file. If you want to write tensors of higher ranks in the summary file, use the native `tf.summary` of TensorFlow in `model_fn`.

- Using the MXNet engine:
  Add the following code to the script:
```python
batch_end_callbacks.append(mx.contrib.tensorboard.LogMetricsCallback('OBS path'))
```

Precautions

- You will be charged as long as your visualization jobs are in the Running status. We recommend you to stop the visualization jobs when you no longer need them to avoid unnecessary fees. Visualization jobs can be automatically stopped at the specified time. To avoid unnecessary fees, you are advised to enable this function.
- By default, CPU resources are used to run visualization jobs and cannot be changed to other resources.
- The OBS directory you use and ModelArts are in the same region.

Creating a Visualization Job

1. Log in to the ModelArts management console. In the left navigation pane, choose Training Jobs. On the displayed page, click the Visualization Jobs tab.
2. In the upper left corner of the visualization job list, click Create to switch to the Create Visualization Job page.
3. Set Billing Mode to Pay-per-use and Job Type to Visualization Job. Enter the visualization job name and description as required, set the Training Output Path and Auto Stop parameters.
   - Training Output Path: Select the training output path specified when the training job is created.
   - Auto Stop: Enable or disable the auto stop function. A running visualization job will be billed. To avoid unnecessary fees, you can enable the auto stop function to automatically stop the visualization job at the specified time. The options are 1 hour later, 2 hours later, 4 hours later, 6 hours later, and Custom. If you select Custom, you can enter any integer from 1 to 24 hours in the text box on the right.
4. Click **Next**.
5. After confirming the specifications, click **Next**.

In the visualization job list, when the status changes to **Running**, the TensorBoard job has been created. You can click the name of the visualization job to view its details.

**Opening a Visualization Job**

In the visualization job list, click the name of the target visualization job. The **TensorBoard** page is displayed. See **Figure 5-26**. Only the visualization job in the **Running** status can be opened.

**Figure 5-26 TensorBoard page**

**Running or Stopping a Visualization Job**

- **Stopping a visualization job**: You can stop a running visualization job to stop billing when it is no longer needed. In the visualization job list, click **Stop** in the **Operation** column to stop the visualization job.
- **Running a visualization job**: You can run and use a visualization job in the **Canceled** status again. In the visualization job list, click **Run** in the **Operation** column to run the visualization job.
Deleting a Visualization Job

If your visualization job is no longer used, you can delete it to release resources. In the visualization job list, click **Delete** in the **Operation** column to delete the visualization job.

**NOTE**

A deleted visualized job cannot be recovered. You need to create a new visualization job if you want to use it. Exercise caution when performing this operation.

5.10 Auto Search Jobs

5.10.1 Introduction to Auto Search Jobs

Based on the ModelArts platform, the auto search service integrates multiple AutoML technologies, such as automatic data enhancement, automatic hyperparameter search, and neural architecture search, to help you easily obtain AutoML capabilities at the lowest cost, and optimize and accelerate models in actual services.

If you have a set of service model code that defines a complete training, evaluation, and model export process, but you are not satisfied with the model precision, inference speed, and training time, you can slightly modify the code according to **Code Compilation Specifications** to make the auto search service as the upper layer of your service model code. You can call the code for multiple times to change the training model architecture and parameters, train and evaluate the model, and finally obtain a model with higher precision and faster speed.

Types of Auto Search Jobs

ModelArts supports the following search modes:

- **Neural network architecture search (pure NAS search):** In image classification and object detection scenarios, ModelArts provides several ResNet structures. You can replace the original ResNet structures with these structures to improve the accuracy or performance, such a search cost is 4 to 10 times the time consumed by a single network training. In addition, ModelArts supports any architecture search. It provides a self-developed search algorithm (MBNAS) to search for a better architecture in any search space.

- **Hyperparameter search:** ModelArts provides multiple classic hyperparameter search algorithms, which can be used to search for common hyperparameters in deep learning, such as **learning_rate**.

- **Auto data augmentation policy search:** ModelArts provides the optimal pre-processing policy and policy decoder that can be used to replace the data augmentation module in the code. You can also use the ModelArts decoder to search for the optimal augmentation policy that is more suitable for your scenario.

- **Multisearch (any combination of the preceding three items):** NAS search, hyperparameter search, and auto data augmentation policy search can be
used at the same time to obtain better architectures, hyperparameters, and augmentation policies, but the cost is higher.

**Using an Auto Search Job for the First Time**

If you use auto search for the first time, the following information helps you get familiar with the auto search function:

1. First, refer to **Types of Auto Search Jobs** to determine the auto search job that your business needs.
2. Then, view the following examples to learn about auto search jobs of ModelArts based on the sample code and YAML configuration file.
   
   You can refer to **Code Compilation Specifications** and **YAML Configuration File Description** for coding requirements.
   
   - Example: Replacing the Original ResNet-50 with a Better Network Architecture
   - Example: Searching for Hyperparameters Using Classic Hyperparameter Algorithms
   - Example: Searching for Network Architectures Using the MBNAS Algorithm
   - Example: Implementing Auto Data Augmentation Using a Preset Data Augmentation Policy
   - Example: Using Multisearch
3. Prepare the boot script and YAML configuration file of the job. Refer to **Creating an Auto Search Job** to start an auto search job and view the search result.

**Functions of Auto Search Jobs**

<table>
<thead>
<tr>
<th>Function</th>
<th>Sub-Function</th>
<th>Description</th>
<th>Operation Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job management</td>
<td>Auto search job creation</td>
<td>Create an auto search job.</td>
<td><strong>Creating an Auto Search Job</strong></td>
</tr>
<tr>
<td></td>
<td>Job stopping or deletion</td>
<td>Stop a running auto search job, or delete unnecessary auto search jobs. The operation procedure is similar to that of a training job. For details, see the related training job guide.</td>
<td><strong>Stopping or Deleting a Job</strong></td>
</tr>
<tr>
<td>Function</td>
<td>Sub-Function</td>
<td>Description</td>
<td>Operation Guide</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>Job version management</td>
<td></td>
<td>ModelArts allows you to manage job versions to effectively train your model after each tuning. Specifically, ModelArts generates a version each time when a training is performed. You can compare different versions to quickly get the difference. The operation procedure is similar to that of a training job. For details, see the related training job guide.</td>
<td>Managing Training Job Versions</td>
</tr>
<tr>
<td>Job details viewing</td>
<td></td>
<td>After a training job finishes, in addition to managing job versions, you can also check whether the job is satisfactory by viewing the job details, logs, and search results. The operation procedure is similar to that of a training job. For details, see the related training job guide.</td>
<td>Viewing Job Details</td>
</tr>
<tr>
<td>Job parameter management</td>
<td></td>
<td>You can store the parameter settings in ModelArts during job creation so that you can use the stored settings to create follow-up jobs, which makes job creation more efficient. The operation procedure is similar to that of a training job. For details, see the related training job guide.</td>
<td>Managing Job Parameters</td>
</tr>
<tr>
<td>Encoding rules</td>
<td>Code compilation specifications</td>
<td>You need to slightly modify your existing model training code based on the code compilation specifications of the auto search service so that the upper-layer search service can successfully call the user code for training and evaluation.</td>
<td>Code Compilation Specifications</td>
</tr>
<tr>
<td>Function</td>
<td>Sub-Function</td>
<td>Description</td>
<td>Operation Guide</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>YAML configuration file description</td>
<td>To use the hyperparameter search capability to perform multisearch, you need to provide a YAML configuration file to configure the control information required by hyperparameter searches.</td>
<td>YAML Configuration File Description</td>
<td></td>
</tr>
<tr>
<td>Examples</td>
<td>Replacing the original ResNet50 with a better network architecture</td>
<td>Use the classification task of ResNet 50 on the MNIST dataset as an example to describe how to prepare data, download code, modify code, create a job, and view search results in an end-to-end manner.</td>
<td>Example: Replacing the Original ResNet-50 with a Better Network Architecture</td>
</tr>
<tr>
<td></td>
<td>Searching for hyperparameters using classic hyperparameter algorithms</td>
<td>Use the hyperparameter algorithm to optimize the black box function.</td>
<td>Example: Searching for Hyperparameters Using Classic Hyperparameter Algorithms</td>
</tr>
<tr>
<td></td>
<td>Searching for network architectures using the MBNAS algorithm</td>
<td>Use the MBNAS algorithm with the simulated data similar to MNIST.</td>
<td>Example: Searching for Network Architectures Using the MBNAS Algorithm</td>
</tr>
<tr>
<td></td>
<td>Implementing auto data augmentation using a built-in data augmentation policy</td>
<td>Use the classification task of ResNet50 on the MNIST dataset as an example to describe how to use the data augmentation policy.</td>
<td>Example: Implementing Auto Data Augmentation Using a Preset Data Augmentation Policy</td>
</tr>
<tr>
<td></td>
<td>Multisearch</td>
<td>Use the classification task of ResNet50 on the MNIST dataset as an example to describe how to use the multisearch function.</td>
<td>Example: Using Multisearch</td>
</tr>
</tbody>
</table>

### 5.10.2 Creating an Auto Search Job
Prerequisites

- Data has been prepared. Specifically, you have created an available dataset in ModelArts, or you have uploaded the dataset used for the auto search job to the OBS directory.
- You have prepared the code and configuration file for the auto search job and uploaded them to the OBS directory. For details about the AutoSearch code compilation specifications, see Code Compilation Specifications. For details about the compilation specifications of the YAML configuration file, see YAML Configuration File Description.
- At least two empty folders have been created on OBS for storing the search results and job logs.
- The account is not in arrears because resources are consumed when training jobs are running.
- The OBS directory you use and ModelArts are in the same region.

Background

- The AI engine for auto search jobs can only be AutoSearch-python3.6. AutoSearch-python3.6 is an engine provided by ModelArts, with the TensorFlow 1.13.1 or PyTorch 1.0.0 engine preset. You need to use the APIs of TensorFlow and PyTorch in the right version when compiling code for an auto search job.

Creating an Auto Search Job

1. Log in to the ModelArts management console. In the left navigation pane, choose Training Management > Auto Search Jobs. The Auto Search Jobs page is displayed.
2. In the upper left corner of the auto search job list, click Create to switch to the Create Auto Search Job page.
3. Set related parameters and click Next.
   a. Enter basic information, including Billing Mode, Name, Version, and Description. Currently, Billing Mode supports only Pay-per-use. The Version is automatically generated and named by the system in the ascending order of V001, V002, and so on. You cannot manually modify it. Specify Name and Description according to actual requirements.

   Figure 5-27 Basic information about an auto search job

   ![Figure 5-27 Basic information about an auto search job](image)

   b. Set the auto search job parameters, including the data source, algorithm source, and more. For details, see Table 5-38.
**Figure 5-28** Setting job parameters

![Figure 5-28 Setting job parameters](image)

**Table 5-38** Auto search job parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-Click Configuration</td>
<td>If you have saved job parameter configurations in ModelArts, click <strong>One-Click Configuration</strong> and select an existing job parameter configuration as prompted to quickly complete parameter setting for the job.</td>
</tr>
<tr>
<td>Data Source (Dataset)</td>
<td>Select an available dataset and its version from the ModelArts <strong>Data Management</strong> module.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Dataset</strong>: Select an existing dataset from the drop-down list. If no dataset is available in ModelArts, no result will be displayed in the drop-down list.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Version</strong>: Select a version according to the <strong>Dataset</strong> setting.</td>
</tr>
<tr>
<td>Data Source (Data path)</td>
<td>Select the training data from your OBS bucket. On the right of the <strong>Data path</strong> text box, click <strong>Select</strong>. In the dialog box that is displayed, select an OBS folder for storing data.</td>
</tr>
<tr>
<td>AI Engine</td>
<td>The AI engine for auto search jobs can be <strong>AutoSearch-python3.6</strong> or <strong>Autosearch-Ascend-Powered</strong>. The latter is used to search for resources on Ascend 910. By default, the common Python 3.6 environment is used to compile code.</td>
</tr>
<tr>
<td>Select Storage Path</td>
<td>Select an OBS path for storing the AutoSearch code file. Set this parameter to the directory where the <strong>Boot File</strong> is located.</td>
</tr>
<tr>
<td>Boot File</td>
<td>Select the boot file of the AutoSearch code, whose name ends with <strong>.py</strong>.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Search Result Export Path</td>
<td>Select a path for storing job output results. You are advised to select an empty directory. Do not select the directory used for storing the dataset for <strong>Search Result Export Path</strong>.</td>
</tr>
<tr>
<td>Running Parameter</td>
<td>Set the key running parameters in code. Ensure that the parameter names are the same as those in code. The <strong>AutoSearch-python3.6</strong> engine has many built-in running parameters, as listed in <strong>Table 5-41</strong>. These parameters are automatically sent to the service when the job is started. Set mandatory parameters during job creation. For custom parameters in the code, you also need to add the running parameters and their values here for job running. For example, <code>max_steps = 10</code>, where <code>max_steps</code> is a passing parameter in code.</td>
</tr>
<tr>
<td>Select Job Log Path</td>
<td>Select a path for storing log files generated during job running. Click <strong>Select</strong> to select a directory from OBS as the log output path. You can also click <strong>Clear</strong> to delete the configured path.</td>
</tr>
</tbody>
</table>

c. Select resources used for the auto search job. Resources used for auto search jobs are limited. Only part of resource flavors are provided here. Resource-related parameters have been set based on the supported flavors and do not need to be modified. For details, see **Table 5-39**.

**Figure 5-29** Selecting resources

<table>
<thead>
<tr>
<th>Resource Pool</th>
<th>Public resource pools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>GPU</td>
</tr>
<tr>
<td>Specifications</td>
<td>CPU: 8 vCPUs 64 GiB GPU: nvidia-v100 32 GiB</td>
</tr>
<tr>
<td>Compute Nodes</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 5-39** Resource parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Pool</td>
<td>Select resource pools for the job. Currently, only public resource pools can be selected for auto search jobs. <strong>AutoSearch-python3.6</strong> supports only GPU specifications (<strong>CPU: 8 vCPUs 64 GiB GPU: nvidia-v100 32 GiB</strong>), and <strong>Autosearch-Ascend-Powered</strong> supports only NPU specifications.</td>
</tr>
</tbody>
</table>
**Parameter** | **Description**
--- | ---
Compute Nodes | Set the number of compute nodes. If the number of nodes is set to 1, the background computing is in a single-node mode. Currently, only the single-node mode is supported.

d. Configure the subscription function and set whether to save the parameter settings of the auto search job.

**Figure 5-30** Configuring the subscription function

![Figure 5-30 Configuring the subscription function](image)

**Table 5-40** Parameters related to the subscription function and parameter configuration saving

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Notification | Select the resource pool status to be monitored from the event list, and SMN sends a notification message when the event occurs. This parameter is optional. You can choose whether to enable subscription based on actual requirements. If you enable subscription, set the following parameters as required:

- **Topic**: indicates the topic name. You can refer to [Create Topic](#) to create a topic on the SMN console.
- **Event**: indicates the event to be subscribed to. The options are **OnJobRunning**, **OnJobSucceeded**, and **OnJobFailed**, indicating that training is in progress, successful, and failed, respectively. |
If you select this option, the parameter settings of the current job will be saved to facilitate subsequent job creation.

Select **Save Training Parameters** and specify **Configuration Name** and **Description**. After a training job is created, you can switch to the **Job Parameters** tab page to view your saved job parameter configuration. For details, see **Managing Job Parameters**.

e. After setting the parameters, click **Next**.

4. On the displayed **Confirm** page, confirm that the information is correct and click **Next**. Generally, auto search jobs run for a period of time, which may be several minutes or tens of minutes depending on the amount of your selected data and resources.

**NOTE**

The auto search job starts immediately after being created.

You can switch to the auto search job list to view the job's basic information. The **Status** of the newly created job is **Initializing**. If the status changes to **Successful**, the job ends and the output result is stored in the location specified by **Search Result Export Path**. If the **Status** of the job changes to **Running failed**, click the job name to enter the job details page, and view the job logs to troubleshoot faults.

### Preset Running Parameters and API Reference

**Table 5-41** Supported running parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_path</td>
<td>Yes</td>
<td>string</td>
<td>Configuration file path (OBS path) of an auto search job. The OBS path must start with <code>obs://</code>.</td>
</tr>
<tr>
<td>train_url</td>
<td>Yes</td>
<td>string</td>
<td>Path for saving the search result. If you have set the <strong>Search Result Export Path</strong> parameter, the system automatically generates this parameter and its value.</td>
</tr>
</tbody>
</table>
## Table 5-42 API reference

<table>
<thead>
<tr>
<th>Search Space</th>
<th>ModelArts AutoSearch API</th>
<th>Input Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reporter</td>
<td>autosearch.reporter</td>
<td>Any metric, such as accuracy and latency</td>
<td>Example: autosearch.reporter(accuracy=foo, loss=bar) For details, see the code in each example.</td>
</tr>
<tr>
<td>config</td>
<td>autosearch.config</td>
<td>-</td>
<td>Parameters passed by the AutoSearch framework to the training code</td>
</tr>
</tbody>
</table>

## Follow-Up Procedure

For auto search jobs, you can manage versions, stop or delete jobs, view job details, and manage job parameters. The operations are similar to those of training jobs. For details, see the operation guide of the training jobs. Different from training jobs, auto search jobs do not support creation and management of visualization jobs.

- **Stopping or deleting a job**: The procedure is similar to that of a training job. For details, see the operation guide in Stopping or Deleting a Job.
- **Managing job versions**: The procedure is similar to that of a training job. For details, see the operation guide in Managing Training Job Versions.
- **Viewing job details**: The procedure is similar to that of a training job. For details, see the operation guide in Viewing Job Details.
- **Managing job parameters**: The procedure is similar to that of a training job. For details, see the operation guide in Managing Job Parameters.

## 5.10.3 Code Compilation Specifications

### Precautions for Code Compilation

Pay attention to the following when using user code:

- ModelArts passes `trial_id` as the unique identifier for running the user code once. Ensure that the unique identifier is used to save the file to avoid file read/write conflicts when multiple tasks are concurrently executed.
- In the boot code, a method named `train` needs to be provided, so that ModelArts can call the method to perform complete training, evaluation, and model export. If training and evaluation are separated, you can provide a method named `eval` in the code as the entry for model evaluation and export. The AutoSearch framework calls `eval` in the same process after `train` is executed. The `train` and `eval` methods do not have input parameters.
- Note the following when using `argparse`: ModelArts passes extra parameters when running user code. Therefore, you need to use non-exclusive input parameters to parse APIs. For example, use `parse_known_args` of `argparse` to replace `parse_args`. 
Using autosearch.config

The search space defined in YAML is processed by the search algorithm and sent to users in the dict format. You can obtain the search space from the autosearch.config file in the user code.

Generally, you can obtain the configuration corresponding to this training using autosearch.config[‘key’], for example:

```
search_space:
  - type: discrete
    params:
      - name: resnet50
        values: ["1-1111111-2111121111-211111", 
                   "1-1112-1111111111121-11111112111", 
               ]
```

That is, you can obtain the configuration used in this training by calling autosearch.config[‘resnet50’]. In general, the architecture code in config may vary according to the search algorithm. For details, see Example: Replacing the Original ResNet-50 with a Better Network Architecture, Example: Searching for Hyperparameters Using Classic Hyperparameter Algorithms, Example: Searching for Network Architectures Using the MBNAS Algorithm, Example: Implementing Auto Data Augmentation Using a Preset Data Augmentation Policy, and Example: Using Multisearch.

Using autosearch.reporter

You can use autosearch.reporter to feed back training results.

Generally, a search algorithm requires a reward_attr. The larger the reward_attr value, the better. In addition, the search algorithm needs to support mathematical expressions to facilitate multi-object search.

```
search_algorithm:
  type: xxx
  reward_attr: accuracy - 0.1 * latency
```

The two metrics need to be reported to the AutoSearch framework in the training code.

```
autosearch.reporter(accuracy=foo, latency=bar)
```

The search algorithm calculates the reward based on the reported result and updates the algorithm status. Additional information you report will not be used.

Modifying Reporter Without Modifying Code

In addition to using autosearch.reporter to feed back the training result, you can analyze the printed logs to automatically obtain the feedback information. After specifying reward_attr, you need to configure a regular expression pointing to metrics in reward_attr in the YAML configuration file.

For example, if the result information is as follows:

```
I can see atdout in screen
result=-1.000000;
```

The YAML configuration file as follows:

```
search_algorithm:
  type: anneal_search
```
max_concurrent: 2
reward_attr: result
report_keys:
  - name: result
    regex: (?<=result=).+(?=;)
save_model_count: 3
num_samples: 6
mode: max

The system uses the regular expression to search for the corresponding metrics in the printed information and automatically provides feedback. In this way, you do not need to modify the service code or maintain additional code.

Using a Preset Decoder

A decoder converts the input code (character string) into the code of an executable algorithm framework (such as TensorFlow or PyTorch). ModelArts provides several types of decoders. You can use these decoders in the code module to greatly reduce your workload. You do not need to modularize your code or parse network architectures or augmentation policies in the code. You only need to embed the preset decoder code.

- ResNet50 decoder for NAS
  The search space corresponding to this preset decoder is used to search for the number of blocks in each stage of ResNet and search for the block position when channels are doubled. The encoding character string format of the backbone architecture is 111-2111-211111-211, where - is used to separate each downsampling phase by using different resolutions, 1 indicates a regular block when channels are not changed, and 2 indicates a block when the basic channels are doubled. The character string indicates a standard ResNet50 architecture. Therefore, each stage can have any number of 1s, and a total of three 2s are required.
  Other ResNet decoders are similar to the ResNet50 decoder.

- Decoder for data augmentation
  For each of the NumPy, TensorFlow, and PyTorch frameworks or libraries (commonly used on the preprocessing node), a data augmentation decoder is provided to decode character strings into data augmentation operations on the corresponding frameworks or libraries.

<table>
<thead>
<tr>
<th>Decoder Name</th>
<th>Supported Engine</th>
<th>ModelArts AutoSearch API</th>
<th>Input Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50 Decoder</td>
<td>TensorFlow</td>
<td>autosearch.client.nas.backbone.resnet.ResNet50</td>
<td>inputs</td>
<td>Input data tensor</td>
</tr>
<tr>
<td>Decoder Name</td>
<td>Supported Engine</td>
<td>ModelArts AutoSearch API</td>
<td>Input Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>--------------------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>include_top</td>
<td></td>
</tr>
</tbody>
</table>
|              |                |                          |                | ● If this parameter is set to True, outputs of all stages are returned.  
|              |                |                          |                | ● If this parameter is set to False, only the output of the last stage is returned.  
|              |                |                          | mode           |             |
|              |                |                          |                | ● If this parameter is set to eval, bn layer parameters are not trained.  
|              |                |                          |                | ● If this parameter is set to Train, bn layer parameters are trained.  
|              |                |                          | data_format    | Tensor data format. The NHWC and NCHW formats are supported. The default format is NHWC.  
|              |                |                          | load_weight    |             |
|              |                |                          |                | ● If this parameter is set to True, the model parameters trained on ImageNet are preloaded. Currently, only ResNet-50 is supported.  
|              |                |                          |                | ● If this parameter is set to False, the model training starts from the beginning.  
| Tensor Flow | autosearch.client.nas.backbone.resnet.ResNet18 | - | The input parameters are the same as those of the autosearch.client.nas.backbone.resnet.ResNet50 API.  
|              | autosearch.client.nas.backbone.resnet.ResNet34 | - | The input parameters are the same as those of the autosearch.client.nas.backbone.resnet.ResNet50 API.  
|              | autosearch.client.nas.backbone.resnet.ResNet101 | - | The input parameters are the same as those of the autosearch.client.nas.backbone.resnet.ResNet50 API.  


<table>
<thead>
<tr>
<th>Decoder Name</th>
<th>Supported Engine</th>
<th>ModelArts AutoSearch API</th>
<th>Input Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data augmentation decoder</td>
<td>NumPy</td>
<td>autosearch.client.augmentoffline_search.preprocessord.arrayBuilder.ImageClassificationNdarrayBuilder</td>
<td>search_space_type</td>
<td>Currently, only value offline is supported.</td>
</tr>
<tr>
<td>Data augmentation decoder</td>
<td>TensorFlow</td>
<td>autosearch.client.augmentoffline_search.preprocessord.arrayBuilder.ImageClassificationTensorflowBuilder</td>
<td>search_space_type</td>
<td>Currently, only value offline is supported.</td>
</tr>
<tr>
<td>Data augmentation decoder</td>
<td>PyTorch</td>
<td>autosearch.client.augmentoffline_search.preprocessord.arrayBuilder.ImageClassificationPytorchBuilder</td>
<td>search_space_type</td>
<td>Currently, only value offline is supported.</td>
</tr>
</tbody>
</table>

**Built-in Environment Information Reference**

*Table 5-44* describes the built-in environment of ModelArts for auto search jobs. The environment version cannot be modified. You can refer to it when using auto search jobs.

**Table 5-44** Built-in environment information

<table>
<thead>
<tr>
<th>Module</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>9.0</td>
</tr>
<tr>
<td>Python</td>
<td>3.6</td>
</tr>
<tr>
<td>TensorFlow-GPU</td>
<td>1.12.0</td>
</tr>
<tr>
<td>PyTorch</td>
<td>1.0.0</td>
</tr>
<tr>
<td>ray</td>
<td>0.8.0</td>
</tr>
<tr>
<td>Keras</td>
<td>2.3.1</td>
</tr>
<tr>
<td>Pillow</td>
<td>6.2.0</td>
</tr>
<tr>
<td>NumPy</td>
<td>1.18.1</td>
</tr>
<tr>
<td>scipy</td>
<td>1.3.2</td>
</tr>
</tbody>
</table>
5.10.4 YAML Configuration File Description

ModelArts stores all parameters required for running the AutoSearch framework in the YAML file, which is mandatory for running the framework. A proper YAML file is the prerequisite for successful search.

YAML File Configuration Example

The following example is used only to help you understand the YAML file. For more examples, see Example: Replacing the Original ResNet-50 with a Better Network Architecture, Example: Searching for Hyperparameters Using Classic Hyperparameter Algorithms, Example: Searching for Network Architectures Using the MBNAS Algorithm, Example: Implementing Auto Data Augmentation Using a Preset Data Augmentation Policy, and Example: Using Multisearch.

general:
   gpu_per_instance: 1

search_space:
   - type: discrete
     params:
       - name: resnet50
         values: 
           

search_algorithm:
   type: grid_search
   reward_attr: mean_accuracy

scheduler:
   type: FIFOScheduler

Overview of YAML Configuration Files

As shown in the preceding example, the YAML configuration consists of four parts:

- Common configuration
- Search space configuration
- Search algorithm configuration
- (Optional) Scheduler configuration

The common configuration is used to configure the resource information required for a single training. The scheduler configuration is optional. The search space and search algorithm configurations are the core of the YAML configuration.

For NAS, hyperparameter, and auto data augmentation scenarios described in Introduction to Auto Search Jobs, the difference lies in the search space and search algorithm. You can define a search space for hyperparameter search and
select a suitable algorithm to perform a hyperparameter search. Similarly, you can define a search space for network architecture search and select a suitable algorithm to perform a NAS search. If multiple configuration files are submitted in a task, the system performs multisearch based on the YAML files in sequence.

**Common Configuration**

Common configuration items are related to resource consumption during the running of the distributed framework. Table 5-45 lists the common configuration items.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu_per_instance</td>
<td>Number of CPUs occupied by a worker</td>
</tr>
<tr>
<td>gpu_per_instance</td>
<td>Number of GPUs occupied by a worker</td>
</tr>
<tr>
<td>npu_per_instance</td>
<td>Number of NPUs occupied by a worker</td>
</tr>
<tr>
<td>instance_per_trial</td>
<td>Number of workers in a trial. The default value is 1. Multiple nodes are not supported. Therefore, this parameter can be set only to 1.</td>
</tr>
</tbody>
</table>

Example configuration:

If the resources consumed by the original training script are a single node and two CPUs/GPUs, configure the resources as shown in the following command output.

When this configuration is used, if a single node with eight CPUs/GPUs is used to execute an auto search job, and the number of trials executed concurrently exceeds four \((8/2 = 4)\), the excessive trials will wait for execution in the auto search job.

```yaml
general:
  gpu_per_instance: 2
```

**Search Space Configuration**

A search space usually contains multiple variables, each of which has its own type (discrete, continuous) and value range. YAML is abstracted in this way. The following is a configuration example:

```yaml
search_space:
  - params:
    - type: discrete_param
      name: filter_size
      values: [3, 5, 7]
    - type: discrete_param
      name: activation
      values: [relu, sigmoid]
    - type: continuous_param
      name: blocks
      start: 2
      stop: 4
      num: 3
```
ModelArts provides two basic variables.

- **Discrete** (*discrete_param*): The discrete type is very simple. You only need to specify the name and value, for example, `filter_size` and `activation` in the preceding configuration.

- **Continuous** (*continuous_param*): The value range of a continuous variable must be specified, from `start` to `stop`.

In addition, some algorithms support only discrete variables. Therefore, if a continuous variable is entered, it needs to be automatically split into discrete points. Therefore, you need to tell the system how to split a continuous variable. ModelArts provides two methods, corresponding to `linspace` and `logspace` of NumPy. If you are familiar with the two APIs of NumPy, you can easily get started.

- If `base` is not specified, linear splitting is used, which is similar to the functions of `np.linspace`. For example, `blocks` in the example indicates that three values are selected from the range from 2 to 4, that is, 2, 3, 4.

- If `base` is specified, the values are obtained by logarithm, which is similar to the functions of `np.logspace`. That is, `num` values are obtained from `base ** start` to `base ** end`. In the example, `learning_rate` indicates the five values from 10 to the power of negative 5 to 10 to the power of negative 1.

### Search Space Configuration (Simplified Format)

To facilitate configuration, ModelArts provides three additional search space types:

- **Discrete search space**: All parameters in the search space are discrete. After a search space is set as a discrete search space, there is no need to configure the parameter type for all parameters in the search space.
  ```yaml
  - type: discrete
    params:
      - name: resnet50
        values: ["1-11111111-21111211111-21111111", 
               "1-11111211-1111111111-21111111", 
               "1-1111111211-1111111111", 
               "1-1111111111-2111111111", 
               "11-111111111111-1111111111", 
               "1-111111111111-1111111111", 
               "1-111111111111-1111111111", 
               "111-211111111111-1111111111", 
               "111-211111111111-1111111111"]
  ```

- **Continuous search space**: All parameters in the search space are continuous. After a search space is set as a continuous search space, there is no need to configure the parameter type for all parameters in the search space.
  ```yaml
  - type: continuous
    params:
      - name: learning_rate
        start: 0.01
        stop: 0.1
      - name: weight_decay
  ```
• Repetitive discrete space: This search space is abstracted to handle the different value ranges of the same parameter at different positions in NAS search.

The type of a repetitive discrete space is `repeat_discrete`, and the unique attribute `repeat` indicates the number of repetitions in the search space (for the block-like architecture).

The parameters in a repetitive discrete space are discrete, but the `values_each_block` attribute is added to indicate the value range of the parameter in each block.

```python
definition
  search_space:
    - type: repeat_discrete
      name: mbnas
      repeat: 8
      params:
        - name: block_repeat
          values: [0, 1, 2, 3, 4]
          values_each_block: [1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 2, 3, 4], [0, 1, 2, 3, 4], [1, 2, 3, 4], [0, 1, 2, 3, 4]
        - name: neck_filter_ratio
          values: [1, 1, 2]
        - name: kernel_size
          values: [3, 5, 7]
        - name: filter_ratio
          values: [0.5, 0.75, 1, 1.25, 1.5]
```

### Search Algorithm Configuration

After a search space is defined, a search algorithm needs to be set to define how to use these variables in the search space.

Use `search_algorithm` to configure the algorithm. **Table 5-46** describes the two mandatory parameters for algorithm configuration. In addition to the mandatory parameters, you can set different values for different search algorithms. For details, see the algorithm documentation or algorithm code in **Table 5-47**.

**Table 5-46 Mandatory parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Algorithm type</td>
</tr>
</tbody>
</table>
Parameter Description

reward_attr

All algorithms in AutoSearch aim to maximize the value of **reward_attr**. This parameter supports mathematical expressions. To minimize a metric, set this parameter to a negative value, for example, **-loss**. With this function, you can easily achieve multi-object search, for example, a search meeting the requirements of balancing the precision and speed of a model, using mathematical expressions.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Application Scenario</th>
<th>Whether reward_attr Supports Expressions</th>
<th>Configuration Example</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>random_search</td>
<td>Hyperparameter and NAS</td>
<td>Yes</td>
<td><strong>Searching for Hyperparameters Using Classic Hyperparameter Algorithms</strong></td>
<td>For details, see Table 5-48.</td>
</tr>
<tr>
<td>grid_search</td>
<td>Hyperparameter, NAS, and data augmentation</td>
<td>Yes</td>
<td>• <strong>Searching for Hyperparameters Using Classic Hyperparameter Algorithms</strong></td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <strong>Example: Implementing Auto Data Augmentation Using a Preset Data Augmentation Policy</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <strong>Example: Using Multisearch</strong></td>
<td></td>
</tr>
<tr>
<td>tpe_search</td>
<td>Hyperparameter</td>
<td>No</td>
<td><strong>Searching for Hyperparameters Using Classic Hyperparameter Algorithms</strong></td>
<td>For details, see Table 5-49.</td>
</tr>
<tr>
<td>anneal_search</td>
<td>Hyperparameter</td>
<td>No</td>
<td><strong>Searching for Hyperparameters Using Classic Hyperparameter Algorithms</strong></td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>Application Scenario</td>
<td>Whether reward_attr Supports Expressions</td>
<td>Configuration Example</td>
<td>Parameter</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------</td>
<td>------------------------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bayes_opt_search</td>
<td>Hyperparameter</td>
<td>No</td>
<td>Searching for Hyperparameters Using Classic Hyperparameter Algorithms</td>
<td>For details, see Table 5-50.</td>
</tr>
<tr>
<td>mbnas</td>
<td>NAS</td>
<td>Yes</td>
<td>Using the MBNAS Algorithm</td>
<td></td>
</tr>
</tbody>
</table>

**Table 5-48** random_search algorithm parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repeat</td>
<td>Total number of trials</td>
</tr>
</tbody>
</table>

**Table 5-49** tpe_search, annal_search, and bayes_opt_search algorithm parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_concurrent</td>
<td>Number of trials that are executed at the same time</td>
</tr>
<tr>
<td>num_samples</td>
<td>Total number of trials</td>
</tr>
<tr>
<td>mode</td>
<td>Whether a larger or smaller reward_attr is required. The value can be max or min.</td>
</tr>
</tbody>
</table>

**Table 5-50** mbnas algorithm parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_of_arcs</td>
<td>Number of returned architectures</td>
</tr>
</tbody>
</table>

**Scheduler Configuration**

A search algorithm provides different trials based on the search space. After many trials are provided, some trials can only be executed in a queue due to limited resources. The scheduler provided by ModelArts schedules these trials. The scheduler has the right to determine which trials are preferentially performed or even terminate a running trial in advance.

Currently, AutoSearch supports FIFO Scheduler, MedianStoppingRule, and Huawei-developed ModelBasedScheduler (a prediction model-based scheduler, which can be used with random search, grid search, mbnas, and autoevo).
No additional parameters need to be configured for the FIFO Scheduler and Median Stopping Rule schedulers but additional parameters need to be configured for the ModelBasedScheduler scheduler.

- **FIFO Scheduler configuration**
  scheduler:
  ```
  type: FIFO Scheduler
  ```

- **Median Stopping Rule configuration**
  scheduler:
  ```
  type: Median Stopping Rule
  ```

  Median Stopping Rule terminates some trials that are worse than other trials in advance. The judgment principle is as follows: After receiving the report of several steps, if the best result of the current trial is worse than the median of the running average of the same number of steps of earlier trials, the scheduler will terminate or suspend the trial in advance.

- **Model Based Scheduler configuration**
  scheduler:
  ```
  type: Model Based Scheduler
  grace_period: 5
  ```

  Model Based Scheduler uses the full training data of the previous trial to train a loss curve-based prediction model. After part of epochs are trained in the subsequent trial, the prediction model predicts the final metrics of the trial based on the loss information of the epochs. For the trial whose final metrics are not ideal, the scheduler stops it directly.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grace_period</td>
<td>Number of completed trials before the trained prediction model is used for early stopping. The default value is 20.</td>
</tr>
<tr>
<td>input_ratio</td>
<td>Ratio of data used for prediction. The default value is 0.1, indicating that the first 10% of reported data is used to predict the next 90% of the data and the highest precision.</td>
</tr>
<tr>
<td>validation_ratio</td>
<td>Ratio of the loss curve used as the validation set. It is used to select the optimal prediction model. The default value is 0. The recommended value is 0.2.</td>
</tr>
<tr>
<td>ensemble_models</td>
<td>Number of used prediction models. Integrated learning can be used to train multiple prediction models for joint prediction. The default value is 1.</td>
</tr>
</tbody>
</table>

### 5.10.5 Example: Replacing the Original ResNet-50 with a Better Network Architecture

Use the classification task of ResNet-50 on the MNIST dataset as an example.

#### Preparing Data

ModelArts provides a sample MNIST dataset named Mnist-Data-Set in the public OBS bucket. This example uses this dataset. Perform the following operations to
upload the dataset to your OBS directory, for example, `test-modelarts/dataset-mnist`.

1. **Download** the Mnist-Data-Set dataset to the local PC.
2. Decompress the Mnist-Data-Set.zip file to the Mnist-Data-Set directory on the local PC.
3. Upload all files in the Mnist-Data-Set folder to the test-modelarts/dataset-mnist directory on OBS in batches. For details about how to upload files, see *Uploading a File*.

The following provides content of the Mnist-Data-Set dataset. `.gz` is the compressed package.

**NOTE**

In this example, the `.gz` format is used. Ensure that the four packages of the dataset are uploaded to the OBS directory.

- `t10k-images-idx3-ubyte.gz`: validation set, which contains 10,000 samples
- `t10k-labels-idx1-ubyte.gz`: labels of the validation set, which contains the labels of the 10,000 samples
- `train-images-idx3-ubyte.gz`: training set, which contains 60,000 samples
- `train-labels-idx1-ubyte.gz`: labels of the training set, which contains the labels of the 60,000 samples

**Sample Code**

Assume that you have a TensorFlow code for the image classification task based on the MNIST handwritten digit image dataset using ResNet50. You only need to modify five lines of code to replace the ResNet50 architecture with the auto search job. The following shows the code after modification. Comments are added to indicate the changes.

```python
import argparse
import time
import os
import logging
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import autosearch    # Change 1: Import the AutoSearch package.
from autosearch.client.nas.backbone.resnet import ResNet50    # Change 2: Import the preset ResNet50 module to decode the architecture code into the TensorFlow architecture.

parser = argparse.ArgumentParser()
parser.add_argument("--max_steps", type=int, default=100, help="Number of steps to run trainer.")
parser.add_argument("--data_url", type=str, default="MNIST_data")
parser.add_argument("--learning_rate", type=float, default=0.01, # st2
help="Number of steps to run trainer.")
parser.add_argument("--batch_size", type=int, default=1, help="batch size")
FLAGS, unparsed = parser.parse_known_args()
```

---

**ModelArts User Guide (Senior AI Engineers)**

5 Training Management

Issue 01 (2021-01-15) Copyright © Huawei Technologies Co., Ltd.
logger = logging.getLogger(__name__)

def train():
    if is_cloud():
        FLAGS.data_url = cloud_init(FLAGS.data_url)
    mnist = input_data.read_data_sets(FLAGS.data_url, one_hot=True)
    with tf.Graph().as_default():
        sess = tf.InteractiveSession()

        with tf.name_scope("input"):
            x = tf.placeholder(tf.float32, [None, 784], name="x-input")
            y_ = tf.placeholder(tf.int64, [None, 10], name="y-input")
            image_shaped_input = tf.reshape(x, [-1, 28, 28, 1])

            y = ResNet50(image_shaped_input, include_top=True, mode="train")  # Change 3: Replace ResNet50 in the original code with the imported ResNet50 decoding module.

        with tf.name_scope("cross_entropy"):
            y = tf.reduce_mean(y, [1, 2])
            y = tf.layers.dense(y, 10)

        with tf.name_scope("total"):
            cross_entropy = tf.losses.softmax_cross_entropy(y_, y)

        with tf.name_scope("train"):
            train_step = tf.train.AdamOptimizer(FLAGS.learning_rate).minimize(  # st2
                cross_entropy)

        with tf.name_scope("accuracy"):
            correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
            accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

        tf.global_variables_initializer().run()

        def feed_dict(train):
            if train:
                xs, ys = mnist.train.next_batch(100)
            else:
                xs, ys = mnist.test.images, mnist.test.labels
            return {x: xs, y_: ys}

        max_acc = 0
        latencies = []
        for i in range(FLAGS.max_steps):
            if i % 10 == 0:  # Record summaries and test-set accuracy
                loss, acc = sess.run(
                    [cross_entropy, accuracy], feed_dict=feed_dict(False)
                )
                print("loss at step %s: %s" % (i, loss))
                print("acc step %s: %s" % (i, acc))
                if acc > max_acc:
                    max_acc = acc
                    autosearch.reporter(loss=loss, mean_accuracy=max_acc)    # Change 4: Report the precision to the AutoSearch framework.
            else:
                start = time.time()
                loss, _ = sess.run(
                    [cross_entropy, train_step], feed_dict=feed_dict(True)
                )
                end = time.time()
                if i % 10 != 1:
                    latencies.append(end - start)
                latency = sum(latencies) / len(latencies)
                autosearch.reporter(mean_accuracy=max_acc, latency=latency)    # Change 4: Report the precision to the AutoSearch framework.

        sess.close()

    def is_cloud():
        return True if os.path.exists("/home/work/user-job-dir") else False
def cloud_init(data_url):
    local_data_dir = "/cache/mnist"

    import moxing as mox

    logger.info(
        'Copying from data_url({})" to local path({})'.format(data_url, local_data_dir)
    )
    mox.file.copy_parallel(data_url, local_data_dir)

    return local_data_dir

Compiling the Configuration File

general:
    gpu_per_instance: 1

search_space:
    - type: discrete
      name: resnet50

search_algorithm:
    type: grid_search
    reward_attr: mean_accuracy

scheduler:
    type: FIFOScheduler

Starting a Search Job

Create an auto search job following the instructions in Creating an Auto Search Job. Set the boot file to the sample code file in Sample Code and set config_path to the OBS path of the sample YAML file, for example, obs://bucket_name/config.yaml. After the configuration is complete, submit the job to start the search job.

- The sample code needs to be stored as a .py file, which is the boot script of the search job.
- The YAML configuration file must end with .yaml.
- The boot script and YAML configuration file can be named based on the actual service.
- The boot script and YAML configuration file must be uploaded to OBS in advance, and the OBS bucket must be in the same region as ModelArts.
Figure 5-31 Setting an auto search job

Viewing Search Results

After the auto search job finishes, click the job name to go to the job details page. Then, click the **Search Results** tab to view the search results.

5.10.6 Example: Searching for Hyperparameters Using Classic Hyperparameter Algorithms

This section describes how to use hyperparameter search by optimizing the black box function. In general, the hyperparameter search problem is a black box optimization problem.

Sample Code

```python
import time
import autosearch    # Change 1: Import the AutoSearch package.

def black_box_function(x, y):
    """Function with unknown internals we wish to maximize.
    This is just serving as an example, for all intents and purposes think of the internals of this function, i.e.: the process which generates its output values, as unknown.
    """
    return -(x ** 2) - (y - 1) ** 2 + 1

def train():
    result = black_box_function(autosearch.config['x'], autosearch.config['y'])    # Change 2: Obtain the parameters delivered by the framework.
    time.sleep(0.2)
    autosearch.reporter(result=result)    # Change 3: Send the result to the AutoSearch framework.
```

The search objective of the preceding sample code is to find the maximum value of the black_box_function function.

Compiling the Configuration File

When the Naïve Bayes algorithm is used for optimization, you can configure the YAML file as follows:
Starting a Search Job

After uploading the preceding script and YAML file to OBS, you can start the job on the page. Select an existing dataset or an empty OBS directory because no actual data is required. For details about how to select other configurations, see Starting a Search Job in Example: Searching for Hyperparameters Using Classic Hyperparameter Algorithms.

Using Other Hyperparameter Algorithms

ModelArts supports simple random search, grid search, and three other classic hyperparameter search algorithms. To use different hyperparameter algorithms, you only need to modify search_algorithm in the YAML file. For details about the algorithm parameters, see Table 5-48, Table 5-49, and Table 5-50 in YAML Configuration File Description.

- To use random search, configure the parameters as follows:

```
search_algorithm:
  type: random_search
  reward_attr: result
  repeat: 1000
```

- To use grid search, configure the parameters as follows:

```
search_algorithm:
  type: grid_search
  reward_attr: result
```

Grid search traverses all possibilities in the search space by default, and is applicable only to scenarios where the search space is not large.

5.10.7 Example: Searching for Network Architectures Using the MBNAS Algorithm

Model-based Neural Architecture Search (MBNAS) a Huawei-developed NAS algorithm.
The work principle is as follows:

- In the search space, 200 architectures are sampled in a random complementary set manner, and the 200 architectures are trained in parallel, to obtain rewards corresponding to the 200 architectures.
- The 200 pieces of data are used to train an Evaluator, and the Evaluator can predict the reward corresponding to a sampled architecture.
- Based on the preceding Evaluator, a Controller based on reinforcement learning or evolutionary algorithms is trained.
- The trained Controller outputs several excellent architectures.

For convenience, this section uses simulated data similar to MNIST as an example. You can also modify the following code to use the MNIST dataset in **Example: Replacing the Original ResNet-50 with a Better Network Architecture**.

**Sample Code**

In the sample code, comments are added at the end of each line to indicate the changes to the original code.

```python
import argparse
import logging
import time
import tensorflow as tf
from tensorflow import keras
```
import autosearch  # Change 1: Import the AutoSearch package.

parser = argparse.ArgumentParser()
parser.add_argument(
    "--max_steps", type=int, default=10 ** 2, help="Number of steps to run trainer.
"
)
FLAGS, unparsed = parser.parse_known_args()

batch_size = 1
learning_rate = 0.001

def train(config, reporter):
    nas_code_extra = config["nas_code_extra"]  # Change 2: Obtain the parameters delivered by the framework.

    with tf.Graph().as_default():
        sess = tf.InteractiveSession()
        x = tf.ones([batch_size, 784], name="x-input")
        y_true = tf.ones([batch_size, 10], name="y-input", dtype=tf.int64)

        for i, stage in enumerate(nas_code_extra):
            x = keras.layers.Dense(
                units=stage["hidden_units"], activation=stage["activation"],
            )(x)  # Change 3: Construct the model based on the input model parameters.

        y = tf.layers.dense(x, 10)
        cross_entropy = tf.losses.softmax_cross_entropy(y_true, y)
        train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)

        correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_true, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

        tf.global_variables_initializer().run()
        max_acc = 0
        latencies = []

        for i in range(FLAGS.max_steps):
            logging.info("current step: {}").format(i))
            if i % 10 == 0:  # Record summaries and test-set accuracy
                loss, acc = sess.run([cross_entropy, accuracy])
            # print("Accuracy at step %s: %s" % (i, acc))
            if acc > max_acc:
                max_acc = acc
            if i == (FLAGS.max_steps - 1):
                autosearch.reporter(
                    loss=loss, acc=max_acc, mean_loss=loss, done=True
                )  # Change 4: Send precision metrics.
            else:
                autosearch.reporter(loss=loss, acc=acc, mean_loss=loss)  # Same as change 4.
            else:
                start = time.time()
                loss, _ = sess.run([cross_entropy, train_step])
                end = time.time()
                if i % 10 != 1:
                    latencies.append(end - start)
                latency = sum(latencies) / len(latencies)
                print((max_acc, latency)

Compiling the Configuration File

**repeat_discrete** is used together with MBNAS. According to the YAML configuration, the content in the **for** loop in the preceding code will be executed for four times.

general:
gpu_per_instance: 0

search_space:
- type: repeat_discrete
Starting a Search Job

After uploading the script and YAML file in the sample code to OBS, you can start the job on the page. Select an existing dataset or an empty OBS directory because no actual data is required. For details about how to select other configurations, see Starting a Search Job in Example: Searching for Hyperparameters Using Classic Hyperparameter Algorithms.

5.10.8 Example: Implementing Auto Data Augmentation Using a Preset Data Augmentation Policy

On the CIFAR-10 dataset, the best data augmentation policy is found using the Auto Augment algorithm. This section demonstrates how to use a found data augmentation policy.

Sample Code

This is the code for training the image classification model on MNIST using the ResNet50 network. The additional changes required for using the auto data augmentation policy in the training code are described in the comments.

```python
import argparse
import time

import tensorflow as tf
from autosearch.client.augment.offline_search.preprocessor_builder import (
    ImageClassificationTensorflowBuilder,
)  # Change 1: Import the decoder module.
from autosearch.client.nas.backbone.resnet import ResNet50
from tensorflow.examples.tutorials.mnist import input_data

import autosearch

parser = argparse.ArgumentParser()
parser.add_argument("--max_steps", type=int, default=100, help="Number of steps to run trainer."
)  # Change 2: Define the number of steps.
parser.add_argument("--data_url", type=str, default="MNIST_data")

parser.add_argument("--learning_rate",
    type=float,
    default=0.01,
    help="Number of steps to run trainer."
)
FLAGS, unparsed = parser.parse_known_args()

def train():
    mnist = input_data.read_data_sets(FLAGS.data_url, one_hot=True)
    # Rest of the training code
```
with tf.Graph().as_default():
    sess = tf.Session()
    with tf.name_scope("input"):
        x = tf.placeholder(tf.float32, [None, 784], name="x-input")
        y_ = tf.placeholder(tf.int64, [None, 10], name="y-input")
        image_shaped_input = tf.multiply(x, 255)
        image_shaped_input = tf.cast(image_shaped_input, tf.uint8)
        image_shaped_input = tf.reshape(image_shaped_input, [-1, 28, 28, 3])
        y = tf.reduce_mean(y, [1, 2])
        y = tf.layers.dense(y, 10)
        with tf.name_scope("total"):
            cross_entropy = tf.losses.softmax_cross_entropy(y_, y)

    with tf.name_scope("train"):
        train_step = tf.train.AdamOptimizer(FLAGS.learning_rate).minimize(cross_entropy)

    with tf.name_scope("accuracy"):
        with tf.name_scope("correct_prediction"):
            correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
        with tf.name_scope("accuracy"):
            accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
        with tf.name_scope("total"):
            accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
        tf.global_variables_initializer().run()

    def feed_dict(train):
        if train:
            xs, ys = mnist.train.next_batch(100)
        else:
            xs, ys = mnist.test.next_batch(10000)
        return {x: xs, y_: ys}

    max_acc = 0
    latencies = []
    for i in range(FLAGS.max_steps):
        if i % 10 == 0:  # Record summaries and test-set accuracy
            loss, acc = sess.run([cross_entropy, accuracy], feed_dict=feed_dict(False))
            if acc > max_acc:
                max_acc = acc
        else:
            start = time.time()
            loss_, _ = sess.run([cross_entropy, train_step], feed_dict=feed_dict(True))
            end = time.time()
            latency = end - start
            latencies.append(latency)
        if i % 10 != 1:
            latencies.append(end - start)
        latency = sum(latencies) / len(latencies)
        print("Accuracy at step %s: %s" % (i, acc))
        if max_acc < acc:
            max_acc = acc
        autosearch.reporter(mean_accuracy=acc)  # Same as change 3.
        autosearch.reporter(loss=loss)
        autosearch.reporter(mean_accuracy=acc)  # Change 3: Report the precision to the AutoSearch framework.
        else:
            start = time.time()
            loss, _ = sess.run([cross_entropy, train_step], feed_dict=feed_dict(True))
            end = time.time()
            latency = end - start
            latencies.append(end - start)
        latency = sum(latencies) / len(latencies)
        autosearch.reporter(mean_accuracy=acc, latency=latency)  # Same as change 3.
        sess.close()

    def cloud_init(data_url):
        pass
Because the policy is searched on CIFAR-10, the supported data is in the CIFAR-10 format, that is, three RGB channels. The value range of each pixel is 0–255. The MNIST data is in single-channel mode by default, and the pixel value range is normalized to 0–1. Therefore, in the preceding sample code, there is an additional operation: converting the data into the CIFAR-10 format. For details, see the following code snippet.

```python
image_shaped_input = tf.multiply(x, 255)
image_shaped_input = tf.cast(image_shaped_input, tf.uint8)
image_shaped_input = tf.reshape(image_shaped_input, [-1, 784, 1])
image_shaped_input = tf.concat([image_shaped_input, image_shaped_input, image_shaped_input], axis=2)
image_shaped_input = ImageClassificationTensorflowBuilder("offline")(image_shaped_input)  # Change 2: The decoder module automatically parses the parameters delivered by the framework and converts the parameters into corresponding augmentation operations.
image_shaped_input = tf.cast(image_shaped_input, tf.float32)
image_shaped_input = tf.reshape(image_shaped_input, [-1, 28, 28, 3])
image_shaped_input = tf.multiply(image_shaped_input, 1 / 255.0)
```

### Compiling the Configuration File

**grid_search** is used to transfer parameters to the decoder module embedded in the code. Actually, there is only one policy.

```yaml
general:
  gpu_per_instance: 1

search_space:
  - type: discrete
    params:
      - name: image_classification_auto_augment
        values: [
          ["4-4-3", "6-6-7", "7-3-9", "6-7-9", "1-6-5", "1-5-1", "5-6-7", "7-6-5", "6-3-7", "0-5-8", "0-9-4", "0-5-6", "14-3-5", "1-6-5", "6-0-8", "4-8-8", "14-2-6", "4-8-6", "14-2-6", "0-8-1", "14-4-1", "1-6-5", "6-0-0", "14-5-2", "0-9-5", "6-5-3", "5-7-5", "6-0-2", "14-2-8", "14-1-5", "0-9-4", "1-8-4", "6-0-7", "1-4-7", "14-2-5", "1-7-5", "1-6-8", "4-6-2", "4-3-7", "4-2-4", "0-5-2", "14-7-2", "0-2-0", "1-1-0", "6-9-3", "0-4-1", "1-8-8", "1-7-7", "1-7-7", "14-5-0", "1-3-7", "0-4-8", "6-9-6", "4-2-8", "0-1-5", "6-0-0", "8-2-4", "1-1-1", "1-7-7", "0-6-4", "1-8-2", "0-9-5", "1-5-0", "14-6-6", "1-9-5", "4-7-0", "0-7-3", "1-7-0", "6-5-1", "5-1-7", "5-1-4", "14-6-5", "0-3-9", "8-5-3", "0-9-2", "2-0-3", "14-6-3", "4-2-4", "1-1-4", "1-7-6", "1-3-8", "0-4-3", "14-6-4", "0-7-6", "0-2-9", "6-4-8", "1-1-0", "1-0-5", "1-8-4", "1-0-4", "1-5-5", "0-1-2", "14-5-5", "0-9-5", "0-6-1", "0-7-8", "1-2-0", "0-1-2", "1-6-9", "1-4-4"]

search_algorithm:
  type: grid_search
  reward_attr: mean_accuracy

scheduler:
  type: FIFOScheduler
```

### Starting a Search Job

The MNIST dataset is required in this example. Upload and configure the dataset by following instructions in [Example: Replacing the Original ResNet-50 with a Better Network Architecture](#), upload the Python script and YAML configuration file, and start the search job. For details, see the section of creating an auto search job.
5.10.9 Example: Using Multisearch

Multisearch refers to the implementation of at least two of data augmentation, hyperparameter, and NAS searches at the same time in a job.

AutoSearch implements multisearch in serial mode. Take auto data augmentation, hyperparameter, and NAS searches as an example. First, search for the optimal data augmentation policy. Then, search for the optimal learning rate using the optimal data augmentation policy. Finally, search for the optimal architecture using the optimal data augmentation policy and learning rate.

Scenario

This section describes how to search for the most appropriate data augmentation policy, learning rate, and network architecture in the search space of the specified data augmentation policy, hyperparameter, and ResNet50-like architecture based on the MNIST dataset when BatchSize is set to 100 and only 5000 steps are trained.

In the preceding scenario, the following functions are used:

- A script that can:
  - Parse the data augmentation policy.
  - Parse a NAS code of ResNet50.
  - Parse learning rate parameters on the CLI.
- YAML configuration for searching for auto data augmentation policies
- YAML configuration for searching for learning rate hyperparameters
- YAML configuration for searching for the ResNet50 architecture

Combine the preceding three YAML configurations and code to complete a multisearch.

Sample Code

The complete code is as follows. The following describes the configuration of some main functions in the code.

```python
import argparse
import time
import os
import tensorflow as tf
from autosearch.client.augment.offline_search.preprocessor_builder import (ImageClassificationTensorflowBuilder,
)
from autosearch.client.nas.backbone.resnet import ResNet50
from tensorflow.examples.tutorials.mnist import input_data
import autosearch

parser = argparse.ArgumentParser()
parser.add_argument("--max_steps", type=int, default=100, help="Number of steps to run trainer."
)
parser.add_argument("--data_url", type=str, default="MNIST_data")

parser.add_argument("--learning_rate",

```
```python
def train():
    if is_yundao():
        data_url = yundao_init(FLAGS.data_url)
        mnist = input_data.read_data_sets(data_url, one_hot=True)
        with tf.Graph().as_default():
            sess = tf.InteractiveSession()
            with tf.name_scope("input"):
                x = tf.placeholder(tf.float32, [None, 784], name="x-input")
                y_ = tf.placeholder(tf.int64, [None, 10], name="y-input")
                x_aug = ImageClassificationTensorflowBuilder("offline")(x)
                image_shaped_input = tf.reshape(x_aug, [-1, 28, 28, 1])
                y = ResNet50(image_shaped_input, include_top=True, mode="train")
            with tf.name_scope("cross_entropy"):
                y = tf.reduce_mean(y, [1, 2])
                y = tf.layers.dense(y, 10)
            with tf.name_scope("total"):
                cross_entropy = tf.losses.softmax_cross_entropy(y_, y)
            with tf.name_scope("train"):
                train_step = tf.train.AdamOptimizer(FLAGS.learning_rate).minimize(  # st2
cross_entropy
            )
        with tf.name_scope("accuracy"):
            with tf.name_scope("correct_prediction"):
                correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
            with tf.name_scope("accuracy"):
                accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
        tf.global_variables_initializer().run()
        def feed_dict(train):
            if train:
                xs, ys = mnist.train.next_batch(100)
            else:
                xs, ys = mnist.test.images, mnist.test.labels
            return {x: xs, y_: ys}
        max_acc = 0
        latencys = []
        for i in range(FLAGS.max_steps):
            if i % 10 == 0:  # Record summaries and test-set accuracy
                loss, acc = sess.run([cross_entropy, accuracy], feed_dict=feed_dict(False))
            else:
                start = time.time()
                loss, _ = sess.run([cross_entropy, train_step], feed_dict=feed_dict(True))
                end = time.time()
                if i % 10 != 1:
                    latencys.append(end - start)
                latency = sum(latencys) / len(latencys)
                autosearch.reporter(mean_accuracy=max_acc, latency=latency)
            if acc > max_acc:
                max_acc = acc
                autosearch.reporter(mean_accuracy=acc)
        sess.close()
```

return True if os.path.exists("/home/work/user-job-dir") else False

def yundao_init(data_url):
    local_data_dir = "\cache/mnist"

    import moxing as mox

    mox.file.copy_parallel(data_url, local_data_dir)

    return local_data_dir

● In the import statement, import a module for parsing augmentation policies and a module for parsing NAS architecture code.

    # Parse the augmentation policies and convert the policy code into data augmentation operations. From autosearch.client.augment.offline_search.preprocessor_builder import ImageClassificationTensorflowBuilder
    # Parse NAS architecture code. The architecture code such as 111-2111-211111-2111 is translated into the TensorFlow network architecture. From autosearch.client.nas.backbone.resnet import ResNet50

    For details about the two preset decoder modules used in the code, see Using a Preset Decoder.

● The script provides some configurable command line parameters.

    parser = argparse.ArgumentParser()
    parser.add_argument("--max_steps", type=int, default=100, help="Number of steps to run trainer")
    parser.add_argument(\"--data_dir\", type=str, default="MNIST_data")
    parser.add_argument(\"--learning_rate\", type=float, default=0.01, help="Number of steps to run trainer")

    FLAGS, unparsed = parser.parse_known_args()

● The script defines a train function. The function name must be train. AutoSearch automatically searches for the train function in the script. In the train function, generate an ImageClassificationTensorflowBuilder that is imported previously to perform auto data augmentation, and use a ResNet50 object that is imported previously to connect the input x and output y.

    After the two objects are inserted into the code, the input data augmentation policy can be translated into a data augmentation operation, and the ResNet architecture code is a ResNet network architecture of TensorFlow.

    def train():
        mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)
        with tf.Graph().as_default():
            sess = tf.InteractiveSession()
            with tf.name_scope(\"input\"):
                x = tf.placeholder(tf.float32, \[None, 784\], name=\"x-input\")
                y_ = tf.placeholder(tf.int64, \[None, 10\], name=\"y-input\")

                x_aug = ImageClassificationTensorflowBuilder(\'offline\')(x)

                image_shaped_input = tf.reshape(x_aug, \[-1, 28, 28, 1\])

                y = ResNet50(image_shaped_input, mode="train")

● The current training result needs to be reported at a proper time for AutoSearch to record, and a better attempt is made based on the existing result by the search algorithm. AutoSearch automatically searches for the train function in the script. In the train function, generate an ImageClassificationTensorflowBuilder that is imported previously to perform auto data augmentation, and use a ResNet50 object that is imported previously to connect the input x and output y.

Compiling the Configuration File

● YAML configuration for data augmentation
  - Use the single-node, single-CPU/GPU configuration for a single training.
  - Use the built-in image_classification_auto_augment. For details about the built-in space, see Using a Preset Decoder.
- Use GridSearch as the search algorithm. **mean_accuracy** is used as the metric. A larger value indicates better performance.

```yaml
general:
gpu_per_instance: 1

search_space:
- type: discrete
  params:
    - name: image_classification_auto_augment
      values: [['14-5-5', '14-5-5'],
                ['14-6-6', '14-6-6'],
                ['14-7-7', '14-7-7'],
                ['14-8-8', '14-8-8'],
                ['14-9-9', '14-9-9']

search_algorithm:
type: grid_search
reward_attr: mean_accuracy
```

**YAML configuration for hyperparameter search**

Use the YAML file to search for the learning rate. Try 10 to the power of negative 1, 2, 3, 4, and 5.

```yaml
general:
gpu_per_instance: 1

search_space:
- type: continuous
  params:
    - name: learning_rate
      start: -1
      stop: -5
      num: 5
      base: 10

search_algorithm:
type: grid_search
reward_attr: mean_accuracy
```

**YAML configuration for searching for the NAS architecture**

Based on the YAML configuration, the GridSearch algorithm of AutoSearch traverses the 11 architectures to be provided.

```yaml
general:
gpu_per_instance: 1

search_space:
- type: discrete
  params:
    - name: resnet50
      values: ['1-11111111-211121111-2111111',
                '1-11122111111111121-111111112111',
                '1-11111121-111111112111',
                '1-1111121-111111112111',
                '1-1111121-111111112111',
                '1-11111111-211121111-21111111',
                '1-11111111-21112111-11111111',
                '1-11111111-21112111-11111111',
                '1-11111111-21112111-11111111',
                '1-11111111-21112111-11111111',
                '1-11111111-21112111-11111111',
                '1-11111111-21112111-11111111',
                '1-11111111-21112111-11111111',
                '1-11111111-21112111-11111111',
                '1-11111111-21112111-11111111']

search_algorithm:
type: grid_search
reward_attr: mean_accuracy
scheduler:
type: FiFOScheduler
```
Combine the script of the sample code and the three YAML files in Compiling the Configuration File to start and run AutoSearch.

Starting a Search Job

The MNIST dataset is required in this example. Upload and configure the dataset by following instructions in Example: Replacing the Original ResNet-50 with a Better Network Architecture, upload the Python script and YAML configuration file, and start the search job. For details, see the section of creating an auto search job. If multiple YAML files are configured, set this parameter to the full paths of the YAML files and separate them with commas (,), for example, obs://foo/bar.yaml,obs://bar/foo.yaml.

5.10.10 Example: Using Backbone Replacement to Optimize Built-in Algorithms on Ascend 910 Without Modifying Code

On Ascend 910, ModelArts finds some better structures than the original ResNet-50. This example describes how to use the found structures to optimize the ResNet-50 of the built-in algorithms without modifying code.

Sample Code

The following sample code uses a built-in algorithm to train the flower dataset on the ResNet-50 network.

```python
import argparse
import os
import json
import sys
import moxing as mox
from moxing.tensorflow.builtin_algorithms.algorithm_common import AlgorithmLego
app_url = os.path.dirname(os.path.dirname(__file__))
sys.path.insert(0, os.path.dirname(__file__))
from algo_util import data_util

parser = argparse.ArgumentParser(description='')
parser.add_argument('--task_type', type=str, default=None, help='')
parser.add_argument('--train_url', type=str, default=None, help='')

def main():
    args, _ = parser.parse_known_args()
    flag_dict = data_util.do_preprocess()
    algo = AlgorithmLego()
    algo.add(args.task_type, flag_dict=flag_dict)
    algo.start()

    mox.file.shift("os", "mox")
    with open("{}\metric.json".format(args.train_url)) as f:
        metric = json.load(f)
        accuracy = metric["total_metric"]["total_metric_values"]["accuracy"]
        print("accuracy:%d." % accuracy)

if __name__ == "__main__":
    main()
```
Compiling the Configuration File

Compile the YAML configuration file. For details, see **Modifying Reporter Without Modifying Code** in **CodeCompilationSpecifications** and **Example: Replacing the Original ResNet-50 with a Better Network Architecture**.

```yaml
general:
  cpu_per_instance: 1
  gpu_per_instance: 0
  npu_per_instance: 1

search_space:
  builtin: ResNet50

search_algorithm:
  type: grid_search
  reward_attr: accuracy
  report_keys:
    - name: accuracy
      regex: (?<=accuracy:).+(?=.)
  scheduler:
    type: FIFO Scheduler
```

Starting a Search Job

In this example, you need to prepare a dataset based on the input requirements of the built-in algorithm, upload and configure the dataset following the instructions in **Example: Replacing the Original ResNet-50 with a Better Network Architecture**, upload the Python script and YAML configuration file, and select the Ascend chip-powered search engine based on the section "Creating an Auto Search Job". Then start the search job on the Ascend chip.
6 Model Management

6.1 Model Management Overview

AI model development and optimization require frequent iterations and debugging. Changes in datasets, training code, or parameters may affect the quality of models. If the metadata of the development process cannot be managed in a unified manner, the optimal model may fail to be reproduced.

ModelArts model management allows you to import models generated with all training versions to manage all iterated and debugged models in a unified manner.

Usage Restrictions

- In an automatic learning project, after a model is deployed, the model is automatically uploaded to the model management list. However, models generated by automatic learning cannot be downloaded and can be used only for deployment and rollout.
- Functions such as model import, model version management, and model conversion are available to all users free of charge.

Four Methods of Importing a Model

- **Importing from Trained Models**: You can create a training job on ModelArts and complete model training. After obtaining a satisfactory model, import the model to the Model Management page for model deployment.
- **Importing from a Template**: Because the configurations of models of the same function are similar, ModelArts integrates the configurations of such models into a common template. By using this template, you can easily and quickly import models without compiling the config.json configuration file.
- **Importing from a Container Image**: For AI engines that are not supported by ModelArts, you can import the model you compile to ModelArts using custom images.
- **Importing from OBS**: If you use a frequently-used framework to develop and train a model locally, you can import the model to ModelArts for model deployment.
Model Management Functions

Table 6-1 Model management functions

<table>
<thead>
<tr>
<th>Supported Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importing a Model</td>
<td>Import the trained models to ModelArts for unified management. You can import models using four methods. The following provides the operation guide for each method.</td>
</tr>
<tr>
<td></td>
<td>‧ Importing a Meta Model from a Training Job</td>
</tr>
<tr>
<td></td>
<td>‧ Importing a Meta Model from a Template</td>
</tr>
<tr>
<td></td>
<td>‧ Importing a Meta Model from a Container Image</td>
</tr>
<tr>
<td></td>
<td>‧ Importing a Meta Model from OBS</td>
</tr>
<tr>
<td>Managing Model Versions</td>
<td>To facilitate source tracing and repeated model tuning, ModelArts provides the model version management function. You can manage models based on versions.</td>
</tr>
</tbody>
</table>

6.2 Importing a Model

6.2.1 Importing a Meta Model from a Training Job

You can create a training job on ModelArts and perform training to obtain a satisfactory model. Then import the model to Model Management for unified management. In addition, you can quickly deploy the model as a service.

Background

- If a model generated by the ModelArts training job is used, ensure that the training job has been successfully executed and the model has been stored in the corresponding OBS directory.
- If a model is generated from a training job that uses built-in algorithms, the model can be directly imported to ModelArts without using the inference code and configuration file.
- If a model is generated from a training job that uses a frequently-used framework or custom image, upload the inference code and configuration file to the storage directory of the model by referring to Model Package Specifications.
- The OBS directory you use and ModelArts are in the same region.

Procedure

1. Log in to the ModelArts management console, and choose Model Management > Models in the left navigation pane. The Models page is displayed.
2. Click Import in the upper left corner. The Import page is displayed.
3. On the **Import** page, set related parameters.
   
a. Enter basic information about the model. For details about the parameters, see **Table 6-2**.

### Table 6-2 Parameters of basic model information

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Model name. The value can contain 1 to 64 visible characters, including Chinese characters. Only letters, Chinese characters, digits, hyphens (-), and underscores (_) are allowed.</td>
</tr>
<tr>
<td>Version</td>
<td>Version of the model to be created. For the first import, the default value is <strong>0.0.1</strong>.</td>
</tr>
<tr>
<td>Label</td>
<td>Model label. A maximum of five model labels are supported.</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description of the model</td>
</tr>
</tbody>
</table>

b. Select the meta model source and set related parameters. **Meta Model Source** has four options based on the scenario. For details, see **Four Methods of Importing a Model**. Set **Meta Model Source** to **Training job**. For details about the parameters, see **Table 6-3**.

### Figure 6-1 Setting Meta Model Source to Training job

![Setting Meta Model Source to Training job](image)

### Table 6-3 Parameters of the meta model source

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta Model Source</td>
<td>Select <strong>Training job</strong>, and select a specified training job that has completed training under the current account and its version from the drop-down lists on the right of <strong>Training Job</strong> and <strong>Version</strong> respectively.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Deployment Type</td>
<td>After the model is imported, select the service type that the model is deployed. When deploying a service, you can only deploy the service type selected here. For example, if you only select Real-time services here, you can only deploy real-time services after importing the model.</td>
</tr>
<tr>
<td>Inference Code</td>
<td>Display the model inference code URL. You can copy this URL directly.</td>
</tr>
<tr>
<td>Parameter Configuration</td>
<td>Click on the right to view the input and output parameters of the model.</td>
</tr>
<tr>
<td>Runtime Dependency</td>
<td>List the dependencies of the selected model on the environment. For example, if tensorflow is used and the installation method is pip, the version must be 1.8.0 or later.</td>
</tr>
</tbody>
</table>

c. Set the inference specifications and model description.

- **Min. Inference Specs**: If your model requires certain resources to complete inference, you can configure this parameter to set the minimum specifications required for normal inference after the model is deployed as a service. In later versions, the system will allocate resources based on the inference specifications in service deployment. You can also modify the specifications as required during deployment. Note that the specifications configured here are valid only when real-time services are deployed and the dedicated resource pool is used or when edge services are deployed.

- **Model Description**: To help other model developers better understand and use your models, you are advised to provide model descriptions. Click **Add Model Description** and then set the document name and URL. A maximum of three model descriptions are supported.

![Figure 6-2 Setting the inference specifications and model description](image)

d. Check the information and click **Next**. The model is imported.

In the model list, you can view the imported model and its version. When the model status changes to **Normal**, the model is successfully imported. On this page, you can create new versions, quickly deploy models, publish models to the market, and perform other operations.
Follow-Up Procedure

- **Model Deployment**: On the Models page, click the triangle next to a model name to view all versions of the model. Locate the row that contains the target version, click **Deploy** in the Operation column, and select the deployment type configured when importing the model from the drop-down list. On the Deploy page, set parameters by referring to **Model Deployment Overview**.

### 6.2.2 Importing a Meta Model from a Template

Because the configurations of models of the same function are similar, ModelArts integrates the configurations of such models into a common template. By using this template, you can easily and quickly import models without compiling the **config.json** configuration file.

**Background**

- Because the configurations of models of the same function are similar, ModelArts integrates the configurations of such models into a common template. By using this template, you can easily and quickly import the model. For details about the template, see **Model Template Overview**.
- For details about the supported templates, see **Supported Templates**. For details about the input and output modes of each template, see **Supported Input and Output Modes**.
- Ensure that you have uploaded the model to OBS based on the model package specifications of the corresponding template.
- The OBS directory you use and ModelArts are in the same region.
- Importing and managing models is free of charge and does not generate fees.

**Procedure**

1. Log in to the ModelArts management console, and choose **Model Management > Models** in the left navigation pane. The Models page is displayed.
2. Click **Import** in the upper left corner. The Import page is displayed.
3. On the Import page, set related parameters.
   a. Enter basic information about the model. For details about the parameters, see **Table 6-4**.

**Table 6-4 Parameters of basic model information**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Model name. The value can contain 1 to 64 visible characters, including Chinese characters. Only letters, Chinese characters, digits, hyphens (-), and underscores (_) are allowed.</td>
</tr>
<tr>
<td>Version</td>
<td>Version of the model to be created. For the first import, the default value is <strong>0.0.1</strong>.</td>
</tr>
</tbody>
</table>
b. Select the meta model source and set related parameters. **Meta Model Source** has four options based on the scenario. For details, see [Four Methods of Importing a Model](#). Set **Meta Model Source** to **Template**. For details about the parameters, see **Table 6-5**.

![Figure 6-3 Setting Meta Model Source to Template](image)

**Table 6-5 Parameters of the meta model source**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Template</td>
<td>Select a template from the existing ModelArts template list, such as <strong>TensorFlow-based image classification template</strong>. ModelArts also provides three filter criteria: <strong>Type</strong>, <strong>Engine</strong>, and <strong>Environment</strong>, helping you quickly find the desired template. If the three filter criteria cannot meet your requirements, you can enter keywords to search for the target template. For details about the supported templates, see <strong>Supported Templates</strong>.</td>
</tr>
<tr>
<td>Model Directory</td>
<td>OBS path where a model is saved. Select an OBS path for storing the model based on the input requirements of the selected model template. <strong>NOTE</strong> If a training job is executed for multiple times, different version directories are generated, such as V001 and V002, and the generated models are stored in the <strong>model</strong> folder in different version directories. When selecting model files, specify the <strong>model</strong> folder in the corresponding version directory.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Input and Output Mode      | If the default input and output mode of the selected template can be overwritten, you can select an input and output mode based on the model function or application scenario. 
**Input and Output Mode** is an abstract of the API (apis) in config.json. It describes the interface provided by the model for external inference. An input and output mode describes one or more APIs, and corresponds to a template. 
For example, for TensorFlow-based image classification template, **Input and Output Mode** supports Built-in image processing mode. The input and output mode cannot be modified in the template. Therefore, you can only view but not modify the default input and output mode of the template on the page.
For details about the supported input and output modes, see Supported Input and Output Modes. |
| Deploymen t Type            | After the model is imported, select the service type that the model is deployed. When deploying a service, you can only deploy the service type selected here. For example, if you only select Real-time services here, you can only deploy real-time services after importing the model. |

c. Set the inference specifications and model description.
   
   - **Min. Inference Specs**: If your model requires certain resources to complete inference, you can configure this parameter to set the minimum specifications required for normal inference after the model is deployed as a service. In later versions, the system will allocate resources based on the inference specifications in service deployment. You can also modify the specifications as required during deployment. Note that the specifications configured here are valid only when real-time services are deployed and the dedicated resource pool is used or when edge services are deployed.
   
   - **Model Description**: To help other model developers better understand and use your models, you are advised to provide model descriptions. Click Add Model Description and then set the document name and URL. A maximum of three model descriptions are supported.

Figure 6-4 Setting the inference specifications and model description

![Figure 6-4 Setting the inference specifications and model description](image)

d. Check the information and click Next. The model is imported.
In the model list, you can view the imported model and its version. When the model status changes to **Normal**, the model is successfully imported. On this page, you can create new versions, quickly deploy models, publish models to the market, and perform other operations.

**Follow-Up Procedure**

- **Model Deployment**: On the **Models** page, click the triangle next to a model name to view all versions of the model. Locate the row that contains the target version, click **Deploy** in the **Operation** column, and select the deployment type configured when importing the model from the drop-down list. On the **Deploy** page, set parameters by referring to **Model Deployment Overview**.

**6.2.3 Importing a Meta Model from a Container Image**

For AI engines that are not supported by ModelArts, you can import the model you compile to ModelArts from custom images.

**Prerequisites**

- For details about the specifications and description of custom images, see **Importing a Model Using a Custom Image**.
- The configuration must be provided for a model that you have developed and trained. The file must comply with ModelArts specifications. For details about the specifications, see **Specifications for Compiling the Model Configuration File**. After the compilation is complete, upload the file to the specified OBS directory.
- The OBS directory you use and ModelArts are in the same region.

**Procedure**

1. Log in to the ModelArts management console, and choose **Model Management > Models** in the left navigation pane. The **Models** page is displayed.
2. Click **Import** in the upper left corner. The **Import** page is displayed.
3. On the **Import** page, set related parameters.
   a. Enter basic information about the model. For details about the parameters, see **Table 6-6**.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Model name. The value can contain 1 to 64 visible characters, including Chinese characters. Only letters, Chinese characters, digits, hyphens (-), and underscores (_) are allowed.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Label</td>
<td>Model label. A maximum of five model labels are supported.</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description of the model.</td>
</tr>
</tbody>
</table>

b. Select the meta model source and set related parameters. **Meta Model Source** has four options based on the scenario. For details, see **Four Methods of Importing a Model**. Set **Meta Model Source** to **Container image**. For details about the parameters, see **Table 6-7**.

**Figure 6-5** Setting Meta Model Source to Container image

**Table 6-7** Parameters of the meta model source

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Image Path</td>
<td>Click ( \text{\textparagraph} ) to import the model image from the container image. The model is of the Image type, and you do not need to use <code>swr_location</code> in the configuration file to specify the image location. For details about how to create a custom image, see <strong>Custom Image Overview</strong>. <strong>NOTE</strong> The model image you select will be shared with the administrator, so ensure you have the permission to share the image (images shared with other accounts are unsupported). When you deploy a service, ModelArts deploys the image as an inference service. Ensure that your image can be properly started and provide an inference interface.</td>
</tr>
</tbody>
</table>
### Parameter | Description
---|---
Deployment Type | After the model is imported, select the service type that the model is deployed. When deploying a service, you can only deploy the service type selected here. For example, if you only select **Real-time services** here, you can only deploy real-time services after importing the model.

Configuration File | The **Import from OBS and Edit online** methods are available. The configuration file must comply with certain specifications in **Model Package Specifications**. If you select **Import from OBS**, you need to specify the OBS path for storing the configuration file. You can enable **View Configuration File** to view or edit the configuration file online.

Parameter Configuration | Click on the right to view the input and output parameters of the model.

---

c. Set the inference specifications and model description.

- **Min. Inference Specs**: If your model requires certain resources to complete inference, you can configure this parameter to set the minimum specifications required for normal inference after the model is deployed as a service. In later versions, the system will allocate resources based on the inference specifications in service deployment. You can also modify the specifications as required during deployment. Note that the specifications configured here are valid only when real-time services are deployed and the dedicated resource pool is used or when edge services are deployed.

- **Model Description**: To help other model developers better understand and use your models, you are advised to provide model descriptions. Click **Add Model Description** and then set the document name and URL. A maximum of three model descriptions are supported.

**Figure 6-6** Setting the inference specifications and model description

![Figure 6-6](image)

d. Check the information and click **Next**. The model is imported.

In the model list, you can view the imported model and its version. When the model status changes to **Normal**, the model is successfully imported. On this page, you can create new versions, quickly deploy models, publish models to the market, and perform other operations.
Follow-Up Procedure

- **Model Deployment**: On the Models page, click the triangle next to a model name to view all versions of the model. Locate the row that contains the target version, click Deploy in the Operation column, and select the deployment type configured when importing the model from the drop-down list. On the Deploy page, set parameters by referring to Model Deployment Overview.

6.2.4 Importing a Meta Model from OBS

In scenarios where frequently-used frameworks are used for model development and training, you can import the model to ModelArts for unified management.

Prerequisites

- The model has been developed and trained, and the type and version of the AI engine it uses is supported by ModelArts. Common engines supported by ModelArts and their runtime ranges are described as follows:

<table>
<thead>
<tr>
<th>Engine</th>
<th>Runtime</th>
<th>Precautions</th>
</tr>
</thead>
</table>
| TensorFlow | python3.6, python2.7, tf1.13-python2.7-gpu, tf1.13-python2.7-cpu, tf1.13-python3.6-gpu, tf1.13-python3.6-cpu, tf1.13-python3.7-cpu, tf1.13-python3.7-gpu, tf2.1-python3.7 | - TensorFlow 1.8.0 is used in python2.7 and python3.6.  
  - python3.6, python2.7, and tf2.1-python3.7 indicate that the model can run on the CPU or GPU at the same time. For other runtime values, if the suffix contains cpu or gpu, the model can run only in the CPU or GPU.  
  - The default runtime is python2.7. |
| MXNet | python3.7, python3.6, python2.7 | - MXNet 1.2.1 is used in python2.7, python3.6, and python3.7.  
  - python2.7, python3.6, and python3.7 indicate that the model can run on the CPU or GPU at the same time.  
  - The default runtime is python2.7. |
<table>
<thead>
<tr>
<th>Engine</th>
<th>Runtime</th>
<th>Precautions</th>
</tr>
</thead>
</table>
| Caffe  | python2.7  
|        | python3.6  
|        | python3.7  
|        | python2.7-gpu  
|        | python3.6-gpu  
|        | python3.7-gpu  
|        | python2.7-cpu  
|        | python3.6-cpu  
|        | python3.7-cpu  |  
|        |      |  ● Caffe 1.0.0 is used in python2.7, python3.6, python3.7, python2.7-gpu, python3.6-gpu, python3.7-gpu, python2.7-cpu, python3.7-cpu, and python3.6-cpu.  
|        |      |  ● python 2.7, python3.7, and python 3.6 can only be used to run models applicable to CPU. For other runtime values, if the suffix contains cpu or gpu, the model can run only in the CPU or GPU. You are advised to use the runtime of python2.7-gpu, python3.6-gpu, python3.7-gpu, python2.7-cpu, python3.7-cpu, and python3.6-cpu.  
|        |      |  ● The default runtime is python2.7.  |
| Spark_MLib | python2.7  
|         | python3.6 |  ● Spark_MLib 2.3.2 is used in python2.7 and python3.6.  
|         |      |  ● The default runtime is python2.7.  
|         |      |  ● python 2.7 and python 3.6 can only be used to run models applicable to CPU.  |
| Scikit_Learn | python2.7  
|           | python3.6 |  ● Scikit_Learn 0.18.1 is used in python2.7 and python3.6.  
|           |      |  ● The default runtime is python2.7.  
|           |      |  ● python 2.7 and python 3.6 can only be used to run models applicable to CPU.  |
| XGBoost  | python2.7  
|         | python3.6 |  ● XGBoost 0.80 is used in python2.7 and python3.6.  
|         |      |  ● The default runtime is python2.7.  
|         |      |  ● python 2.7 and python 3.6 can only be used to run models applicable to CPU.  |
The imported model, inference code, and configuration file must comply with the requirements of ModelArts. For details, see Model Package Specifications, Specifications for Compiling the Model Configuration File, and Specifications for Compiling Model Inference Code.

The model package that has completed training, inference code, and configuration file have been uploaded to the OBS directory.

The OBS directory you use and ModelArts are in the same region.

### Procedure

1. Log in to the ModelArts management console, and choose Model Management > Models in the left navigation pane. The Models page is displayed.
2. Click Import in the upper left corner. The Import page is displayed.
3. On the Import page, set related parameters.
   a. Enter basic information about the model. For details about the parameters, see Table 6-9.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Model name. The value can contain 1 to 64 visible characters, including Chinese characters. Only letters, Chinese characters, digits, hyphens (-), and underscores (_) are allowed.</td>
</tr>
<tr>
<td>Version</td>
<td>Version of the model to be created. For the first import, the default value is 0.0.1.</td>
</tr>
<tr>
<td>Label</td>
<td>Model label. A maximum of five model labels are supported.</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description of the model</td>
</tr>
</tbody>
</table>

b. Select the meta model source and set related parameters. Meta Model Source has four options based on the scenario. For details, see Four
**Methods of Importing a Model.** Set **Meta Model Source** to **OBS**. For details about the parameters, see **Table 6-10**.

For the meta model imported from OBS, you need to compile the inference code and configuration file by referring to **Model Package Specifications** and place the inference code and configuration files in the **model** folder storing the meta model. If the selected directory does not contain the corresponding inference code and configuration files, the model cannot be imported.

**Figure 6-7 Setting Meta Model Source to OBS**

Table 6-10 Parameters of the meta model source

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta Model</td>
<td>Select the model storage path. This path is the training output path specified in the training job.</td>
</tr>
<tr>
<td>AI Engine</td>
<td>The corresponding AI engine is automatically associated based on the selected meta model storage path.</td>
</tr>
<tr>
<td>Deployment Type</td>
<td>After the model is imported, select the service type that the model is deployed. When deploying a service, you can only deploy the service type selected here. For example, if you only select <strong>Real-time services</strong> here, you can only deploy real-time services after importing the model.</td>
</tr>
<tr>
<td>Configuration File</td>
<td>By default, the system associates the configuration file stored in OBS. Enable the function to view, edit, or import the model configuration file from OBS.</td>
</tr>
<tr>
<td>Parameter Configuration</td>
<td>Click ➔ on the right to view the input and output parameters of the model.</td>
</tr>
<tr>
<td>Runtime Dependency</td>
<td>List the dependencies of the selected model on the environment. For example, if <strong>tensorflow</strong> is used and the installation method is <strong>pip</strong>, the version must be 1.8.0 or later.</td>
</tr>
</tbody>
</table>

c. Set the inference specifications and model description.
- **Min. Inference Specs**: If your model requires certain resources to complete inference, you can configure this parameter to set the minimum specifications required for normal inference after the model is deployed as a service. In later versions, the system will allocate resources based on the inference specifications in service deployment. You can also modify the specifications as required during deployment. Note that the specifications configured here are valid only when real-time services are deployed and the dedicated resource pool is used or when edge services are deployed.

- **Model Description**: To help other model developers better understand and use your models, you are advised to provide model descriptions. Click **Add Model Description** and then set the document name and URL. A maximum of three model descriptions are supported.

![Figure 6-8 Setting the inference specifications and model description](image)

d. Check the information and click **Next**. The model is imported.

In the model list, you can view the imported model and its version. When the model status changes to **Normal**, the model is successfully imported. On this page, you can create new versions, quickly deploy models, publish models to the market, and perform other operations.

### Follow-Up Procedure

- **Model Deployment**: On the **Models** page, click the triangle next to a model name to view all versions of the model. Locate the row that contains the target version, click **Deploy** in the **Operation** column, and select the deployment type configured when importing the model from the drop-down list. On the **Deploy** page, set parameters by referring to **Model Deployment Overview**.

### 6.3 Managing Model Versions

To facilitate source tracing and repeated model tuning, ModelArts provides the model version management function. You can manage models based on versions.

#### Prerequisites

You have imported a model to ModelArts,

#### Creating a New Version

On the **Model Management > Models** page, click **Create New Version** in the **Operation** column. The **Create New Version** page is displayed. Set related parameters by following the instructions in **Importing Models** and click **Next**.
Deleting a Version

On the Model Management > Models page, click the triangle on the left of the model name to expand a model version list. In the model version list, click Delete in the Operation column to delete the corresponding version.

**NOTE**

A deleted version cannot be recovered. Exercise caution when performing this operation.

6.4 Model Evaluation and Diagnosis

6.4.1 Model Evaluation Overview

After a model is imported to ModelArts, you can evaluate and diagnose the model to determine whether it meets service requirements.

Functions of Model Evaluation

The system provides evaluation metrics for evaluation tasks of different types of models. When the evaluation results are displayed, the system performs detailed evaluation on the models based on different data features to obtain the sensitivity of each data feature to the evaluation metrics and provides optimization suggestions. Users can fully understand the adaptability of models to different data features, so that model optimization can be targeted.

For a training job with a built-in algorithm or evaluation code, you can view the evaluation result after the training is complete. For details, see Viewing the Evaluation Result.

Figure 6-9 Model evaluation

---

Evaluation Jobs and Job Versions

- An evaluation job can contain multiple versions.
- In the same job, you can change the dataset or evaluation code to create a new version.
An evaluation result is generated after an evaluation job version is successfully executed. You can view the evaluation result of each evaluation metric on the evaluation result page under Version Manager.

Model Evaluation Restrictions

- Currently, model evaluation and diagnosis support the following types of models and datasets: image classification, object detection, and semantic segmentation.
- Models for evaluation and diagnosis use TensorFlow or PyTorch as AI engines. Only TF-1.13.1-python3.6, TF-2.1.0-python3.6, and PyTorch-1.4.0-python3.6 can be used to compile evaluation code.
- Only GPU resources are supported. In addition, the resource pool supports only the single-node running mode and does not support the distributed mode.
- Models generated by ExeML do not support model evaluation.
- For the built-in TensorFlow algorithms of ModelArts, you can set parameters to start evaluation after training, or import the models to the model management module and create an evaluation job.

Operations Related to Model Evaluation

- Creating a Model Evaluation Job
- Viewing Evaluation Results
- Evaluation Metrics
- Managing Evaluation Job Versions

6.4.2 Creating a Model Evaluation Job

Choose Model Management > Evaluation/Diagnosis. After compiling the evaluation script, create an evaluation job to evaluate the model. After the evaluation is complete, you can view the metrics of each evaluation job.

Prerequisites

- Ensure that the account is not in arrears because resources are consumed when evaluation jobs are running.
- The model evaluation code has been compiled and uploaded to the OBS directory. The OBS directory you use and ModelArts are in the same region.
- The boot file of the model evaluation code is in .py format.
- You have prepared required data sources. A dataset of the image classification or object detection type has been created on ModelArts, and the dataset has been published. Or, you have uploaded the data used for image classification, object detection, or semantic segmentation to the OBS path.
- For details about how to compile the model evaluation code, see Model Evaluation API and Sample Code for Model Evaluation.

Creating a Job

1. Log in to the ModelArts management console and choose Model Management > Evaluation/Diagnosis from the left navigation pane.
2. On the **Evaluation/Diagnosis** page, click **Create** in the upper left corner. The **Create Model Evaluation Job** page is displayed.

3. On the **Create Model Evaluation Job** page, enter basic project information, job information, and resource pool information.
   a. Set basic information about the job.

   **Table 6-11 Basic information**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Billing Mode</td>
<td>The default value is <strong>Pay-per-use</strong> and cannot be changed.</td>
</tr>
<tr>
<td>Name</td>
<td>Job name, which is used to distinguish jobs.</td>
</tr>
<tr>
<td>Version</td>
<td>It is automatically generated by the system and named in the format of V0001 or V0002. It cannot be modified by users.</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description of the project.</td>
</tr>
</tbody>
</table>

   **Figure 6-10 Basic information**

   ![Billing Mode](image)
   ![Name](image)
   ![Version](image)
   ![Description](image)

   b. Set the job parameters.

   **Table 6-12 Description of model evaluation job parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Source</td>
<td>The available model sources are as follows:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Model Management</strong>: Select an available model and its version from the ModelArts model management list. Models that have been successfully imported under the current account are displayed in the drop-down list.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Model Storage Location</strong>: Upload a model to OBS and select the model for evaluation from the corresponding OBS path.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Job Type</td>
<td>Available values are Image Classification, Object Detection, and Semantic Segmentation. Select an evaluation type based on the application of your model.</td>
</tr>
<tr>
<td>AI Engine</td>
<td>Select the AI engine used by the model evaluation job from the drop-down list on the right. TensorFlow</td>
</tr>
<tr>
<td>Evaluation Code Directory</td>
<td>Path for storing the model evaluation code. The path is an OBS directory. You are advised to upload the model evaluation code to the OBS bucket before creating a job.</td>
</tr>
<tr>
<td>Boot File</td>
<td>After the directory for storing the model evaluation code is set, select the boot file of the model to be evaluated in the directory. The boot file must be in .py format.</td>
</tr>
<tr>
<td>Data Source</td>
<td>There are two types of data sources: Data Management and Data Storage Location. When Job Type is set to Image Classification or Object Detection, you can select data from Data Management or Data Storage Location. When Job Type is set to Semantic Segmentation, you can select data only from Data Storage Location.</td>
</tr>
<tr>
<td></td>
<td>- Data Management: Select a dataset and its version from ModelArts Data Management. The type of the selected dataset must be the same as the evaluation type.</td>
</tr>
<tr>
<td></td>
<td>- Data Storage Location: Select the OBS path where your desired dataset resides.</td>
</tr>
<tr>
<td>Running Parameter</td>
<td>The system sets the model_url and data_url parameters by default based on the preceding configuration. You can click Add Running Parameter to add more parameter settings required for code evaluation.</td>
</tr>
</tbody>
</table>
c. Set the resource pool and its specifications for running the job. Due to the limitations of model evaluation, you can use the default values of resource pool parameters.

**Table 6-13 Resource pool parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Pool</td>
<td>Currently, only public resource pools can be used in the evaluation job.</td>
</tr>
<tr>
<td>Type</td>
<td>Model evaluation requires high-performance resources. Therefore, only the GPU type is supported.</td>
</tr>
<tr>
<td>Specifications</td>
<td>Currently, 8 vCPUs 64 GiB GPUs are supported.</td>
</tr>
<tr>
<td>Compute Nodes</td>
<td>The default value is 1. Currently, only the single-node mode is supported.</td>
</tr>
</tbody>
</table>

**Figure 6-12 Selecting a resource pool and its specifications**

4. Ensure that the settings are correct and click **Next**.
5. On the **Confirm Specifications** page, confirm the parameters of the model evaluation job and click **Submit**.

After the model evaluation job is created, the job starts to run. The running process takes several minutes. On the **Evaluation/Diagnosis** page, if the job status is **Completed**, the evaluation jobs are complete. You can click the job name and select a job version to view the evaluation result.

A public resource pool is used. Therefore, when multiple users submit jobs at the same time, your jobs may be in the **Queuing** state, indicating that they are waiting for idle resources.

**Follow-Up Procedure**

After an evaluation job is created, you can perform the following operations to determine whether the model meets service requirements.

- **Viewing evaluation results**: After a model evaluation job is created and performed, you can view the evaluation result generated by the evaluation job based on the related metrics. For details, see **Viewing Evaluation Results**.
- **Creating a version**: If the result of the first evaluation does not meet service requirements, you can create another job version, change the dataset or evaluation code, and perform the evaluation again. For details, see **Creating a Version**.
- **Deleting an evaluation job**: If an evaluation job is no longer needed, you can delete it to avoid resource waste. On the **Evaluation/Diagnosis** page, locate the row where the target job resides and click **Delete** in the **Operation** column to delete the job. Deleted jobs cannot be recovered. Exercise caution when performing this operation.

**6.4.3 Viewing Evaluation Results**

An evaluation result is generated after each evaluation job is executed. You can view the evaluation result of a single job.

**Viewing Evaluation Results**

1. Log in to the ModelArts management console and choose **Model Management > Evaluation/Diagnosis** from the left navigation pane.
2. On the **Evaluation/Diagnosis** page, select the evaluation job whose result you want to view and click the job name. The **Version Manager** page is displayed.
3. On the **Version Manager** page, select a job version and click the triangle on the left to view the evaluation result of the job version.

The evaluation metrics and optimization suggestions vary depending on the model. Read the results and optimization suggestions provided on the page carefully and optimize your model based on the optimization suggestions.
6.4.4 Evaluation Metrics

The available evaluation metrics are evaluation overview, precision evaluation, sensitivity analysis, computing performance analysis, heatmap, abstract feature distribution, and adversarial analysis. They are suitable for image classification, object detection, and image semantic segmentation scenarios. The evaluation overview, precision evaluation, and sensitivity analysis metrics have specific parameters in the proceeding scenarios, and the computing performance analysis, heatmap, abstract feature distribution, and adversarial analysis metrics are available only in image classification scenarios.

Common Part

Table 6-14 Parameters for evaluation overview

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Metric</td>
<td>It is <strong>Accuracy</strong> for image classification, <strong>mAP</strong> for object detection, and <strong>PA</strong> for image semantic segmentation. For details, see the metric description in specific scenarios.</td>
</tr>
<tr>
<td>Prediction Results</td>
<td>Displays the prediction results, label status, and confidence levels.</td>
</tr>
<tr>
<td>Overall Evaluation</td>
<td>Provides phenomenons and optimization suggestions based on the analysis of prediction results and datasets, and displays the phenomenons and optimization suggestions with a higher priority.</td>
</tr>
</tbody>
</table>

Image Classification

Each column of a confusion matrix represents the actual label statistics, and each row represents the prediction result statistics. The data on the diagonal of the matrix represents all the correct prediction results. Some concepts are used to calculate precision. For example, true positive (TP), false positive (FP), true negative (TN), and false negative (FN) are used for binary classification tasks.
### Table 6-15 Concepts involved in the confusion matrix for image classification

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Actual: Positive</th>
<th>Actual: Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted: Positive</td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>Predicted: Negative</td>
<td>FN</td>
<td>TN</td>
</tr>
<tr>
<td>Total samples</td>
<td>( P = TP + FN )</td>
<td>( N = FP + TN )</td>
</tr>
</tbody>
</table>

### Table 6-16 Evaluation metrics for model classification models

<table>
<thead>
<tr>
<th>Metric</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision Evaluation</td>
<td>Category Distribution</td>
<td>Statistics for the number of different categories of samples.</td>
</tr>
<tr>
<td></td>
<td>Confusion Matrix</td>
<td>For details about the confusion matrix, see Table 6-15.</td>
</tr>
<tr>
<td>Recall (R)</td>
<td></td>
<td>Ratio of the number of correct positive predictions to the total number of positives. A larger value indicates a smaller false negative rate (FNR). The calculation formula is as follows: ( R = TP/(TP + FN) ). That is, the number of correct predictions in a column of the confusion matrix divided by the sum of samples in the column.</td>
</tr>
<tr>
<td>Precision (P)</td>
<td></td>
<td>Ratio of the number of correct positive predictions to the total number of positive predictions. A larger value indicates a smaller false positive rate (FPR). The calculation formula is as follows: ( P = TP/(TP + FP) ). That is, the number of correct predictions in a row of the confusion matrix divided by the sum of samples in the row.</td>
</tr>
<tr>
<td>F1 Score</td>
<td></td>
<td>Harmonic mean of the precision and recall. The formula is as follows: ( F1 = 2 \times P \times R/(P + R) ).</td>
</tr>
<tr>
<td>ROC Curve</td>
<td></td>
<td>The ROC curve is used to draw the true positive rate (TPR, vertical coordinate) and false positive rate (FPR, horizontal coordinate) when different classification thresholds are used. The closer the ROC curve to the upper left corner, the better the classifier performance.</td>
</tr>
<tr>
<td>Sensitivity Analysis</td>
<td>Accuracy in Different Feature Value Ranges</td>
<td>Divide an image into several parts based on the feature values, such as the brightness and clarity, test the precision of these parts, and draw a chart.</td>
</tr>
<tr>
<td>Metric</td>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Feature Distribution</td>
<td></td>
<td>Displays the distribution of image feature values using charts.</td>
</tr>
<tr>
<td>F1 Score</td>
<td></td>
<td>The F1 scores of different types of data in different feature value ranges are displayed to determine the feature value ranges in which the model has a better effect.</td>
</tr>
<tr>
<td>Computing Performance Analysis</td>
<td>Operator Duration Ratio and Parameter Quantity Ratio</td>
<td>Calculate the ratios of various parameters in the network, such as convolution and pooling, as well as the time consumption ratio in a forward process.</td>
</tr>
<tr>
<td>Other Metrics</td>
<td></td>
<td>Includes basic model information such as the GPU usage, time required, model size, total number of parameters, and total computing amount.</td>
</tr>
<tr>
<td>Heatmap</td>
<td>Display Heatmap</td>
<td>Heatmap drawn using the gradcam++ algorithm. The highlighted area indicates the area used by the model to determine the image inference result.</td>
</tr>
<tr>
<td>Metric</td>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Abstract Feature</td>
<td>Feature Distribution</td>
<td>Extracts the convolutional layer output of the layer before the fully connected layer of the basic network for image classification. For example, in a ResNet-50 network, a 1 x 2048 matrix is output for an image. The dimensions of the output are reduced to 2 and are drawn on a 2D scatter chart.</td>
</tr>
<tr>
<td>Adversarial Sample</td>
<td>PSNR</td>
<td>The peak signal-to-noise ratio (PSNR) indicates the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation.</td>
</tr>
<tr>
<td>Evaluation</td>
<td>SSIM</td>
<td>The Structural Similarity Index (SSIM) is used for measuring the similarity between two images. It is often used to compare distortion-free images and distorted images.</td>
</tr>
<tr>
<td></td>
<td>ACAC</td>
<td>Average Confidence of Adversarial Class (ACAC).</td>
</tr>
<tr>
<td></td>
<td>ACTC</td>
<td>Average Confidence of True Class (ACTC). This parameter is used to further evaluate the extent to which the attack deviates from the actual value.</td>
</tr>
<tr>
<td></td>
<td>MR</td>
<td>Proportion of adversarial examples that are classified as incorrect classes or classified as target classes.</td>
</tr>
<tr>
<td></td>
<td>ALD</td>
<td>The Average Lp distortion (ALD) represents the average Lp of successful adversarial examples. A smaller value indicates that adversarial examples are less likely to be detected.</td>
</tr>
<tr>
<td></td>
<td>Others</td>
<td>Similar to the metrics in the precision evaluation.</td>
</tr>
</tbody>
</table>
Computing Performance Analysis supports only built-in TensorFlow-based image classification algorithms. Heatmap, Abstract Feature Distribution, and Adversarial Sample Evaluation support only TensorFlow-based image classification algorithms. To display these metrics, you need to modify the files required for generating the evaluation code. For details, see the image classification part in Sample Code for Model Evaluation.

### Object Detection

<p>| Table 6-17 Evaluation metrics for object detection models |
|---|---|---|
| <strong>Metric</strong> | <strong>Parameter</strong> | <strong>Description</strong> |
| Precision Evaluation | Category Distribution | Statistics for the number of different categories of bounding boxes. |
| Precision-Recall Curve (P-R Curve) |  | Sort samples based on the confidence score of each class, add the samples to the positive samples one by one for prediction, and calculate the precision and recall rates. The curve drawn using this series of precision and recall rates is the P-R curve of the corresponding class. |
| mAP with Different IoUs |  | Calculate the mAP with different IoUs and draw a curve to present the IoU with the highest mAP. The IoU is a threshold used by the NMS to filter an overlapping box that may be predicted as the same object. For details, see Figure 6-14. |
| F1 Scores with Different Confidence Thresholds |  | Calculate the average F1 value under different confidence thresholds, draw a curve, and feed back the threshold with the highest F1 value. |
| False Positive Analysis |  | From the perspective of prediction results, collect statistics on accurate detections, class false positives, background false positives, and position deviations. Draw a pie chart using the proportion of each type of error. For details about error types, see Figure 6-15. |
| False Negative Analysis |  | From the perspective of actual labels, collect statistics on accurate detections, class false positives, background false positives, and position deviations. Draw a pie chart using the proportion of each type of error. For details about error types, see Figure 6-16. |</p>
<table>
<thead>
<tr>
<th>Metric</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity Analysis</td>
<td>Accuracy in Different Feature Value Ranges</td>
<td>It is similar to that for image classification. However, you can select more features related to the target bounding boxes, such as the overlap between the target bounding boxes and the number of target bounding boxes.</td>
</tr>
<tr>
<td>Feature Distribution</td>
<td></td>
<td>It is similar to that for image classification. However, you can select more features related to the target bounding boxes, such as the overlap between the target bounding boxes and the number of target bounding boxes.</td>
</tr>
</tbody>
</table>

**Figure 6-14 IoU calculation**

From the perspective of prediction results, if the IoU of the predicted bounding box and the actual bounding box is greater than 0.5, the predicted bounding box is inconsistent with the actual bounding box, and a class false positive error occurred. If the IoU is greater than 0.1 and less than 0.5, the predicted bounding box is consistent with the actual bounding box, and a position false positive error occurred. If the IoU is less than 0.1, a background false positive error occurred.
Figure 6-15 False positive analysis

From the perspective of the actual bounding box, if the IoU of the actual bounding box and the predicted bounding box is greater than 0.5, the actual bounding box is inconsistent with the predicted bounding box, and a class false negative error occurred. If the IoU is greater than 0.1 and less than 0.5, the actual bounding box is consistent with the predicted bounding box, and a position false negative error occurred. If the IoU is less than 0.1, a background false negative error occurred.
**Image semantic segmentation**

**Table 6-18 Evaluation metrics for image semantic segmentation models**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision Evaluation</td>
<td>Pixel Category Distribution</td>
<td>Statistics for the number of different categories of pixels.</td>
</tr>
</tbody>
</table>
| IoU                   |                                     | It calculates the IoU between each class of prediction result and the label set. You can obtain the mean IoU by averaging the values of each class. The formula for calculating the IoU is as follows:  
\[
\text{IoU} = \frac{P_{ii}}{\sum_{j=0}^{k} \rho_{ij} + \sum_{j=0}^{k} \rho_{ji} - P_{ii}}
\]
Assume that the total number of classes is k + 1, \(P_{ii}\) indicates the number of correct class identifications of the \(i^{th}\) class, and \(p_{ij}\) indicates the number of classes of the \(i^{th}\) class that are identified as the \(j^{th}\) class.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dice Coefficient</td>
<td>The value ranges from 0 to 1. A value closer to 1 indicates a better model. The formula for calculating the Dice coefficient is as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{2^{*}p_{ij}}{k \sum p_{ij} + k \sum p_{ji}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Assume that the total number of classes is $k +1$, $p_{ii}$ indicates the number of correct class identifications of the $i^{th}$ class, and $p_{ij}$ indicates the number of classes of the $i^{th}$ class that are identified as the $j^{th}$ class.</td>
</tr>
<tr>
<td></td>
<td>Confusion Matrix</td>
<td>It is the same as that for image classification except that this confusion matrix is for each pixel instead of each image.</td>
</tr>
<tr>
<td></td>
<td>Sensitivity</td>
<td>Sensitivity Analysis</td>
</tr>
<tr>
<td></td>
<td>Analysis</td>
<td>It is the same as that for image classification except that the evaluation metric is changed from F1 to IoU.</td>
</tr>
</tbody>
</table>

### 6.4.5 Managing Evaluation Job Versions

You can create multiple job versions for an evaluation job to evaluate models based on different dimensions or metrics. You can view, compare, and analyze evaluation results.

**Creating a Version**

1. Log in to the ModelArts management console and choose Model Management > Evaluation/Diagnosis from the left navigation pane.
2. On the Evaluation/Diagnosis page, select the evaluation job for which you want to create a version and click the job name. The Version Manager page is displayed. Click Create Version in the upper right corner. The Create Version page is displayed.
3. On the Create Version page, reset job and resource pool parameters.
   - For details about how to set job parameters, see Creating a Model Evaluation Job. Table 6-12 describes the parameters.
   - For details about how to set resource pool parameters, see Creating a Model Evaluation Job. Table 6-13 describes the parameters.
4. After setting the parameters, click Submit. After the version is created, the system automatically runs the evaluation job. An evaluation result is generated after the evaluation job is successful. You can view the evaluation result of each evaluation metric on the evaluation result page under Version Manager.
### Terminating a Job Version

1. Log in to the ModelArts management console and choose **Model Management > Evaluation/Diagnosis** from the left navigation pane.

2. On the **Evaluation/Diagnosis** page, the basic information about jobs is displayed by default. Select a desired evaluation job and click the job name to go to the **Version Manager** page.

3. On the **Version Manager** page, click **Terminate** in the upper right corner to terminate the job version.

   Only job versions in the **Creating** or **Queuing** state can be terminated. For job versions in the **Completed** state, the **Terminate** button is unavailable.

   After a job version is terminated, it cannot be restarted. You can only create another job version.

### Deleting a Job Version

If you do not need to view the evaluation result of a job version, you can delete the job version to release resources.

1. Log in to the ModelArts management console and choose **Model Management > Evaluation/Diagnosis** from the left navigation pane.

2. On the **Evaluation/Diagnosis** page, select the evaluation job for which you want to manage versions and click the job name. The **Version Manager** page is displayed. In the upper right corner, click **Delete**. In the dialog box that is displayed, confirm the information and click **OK**.

   **NOTE**

   After an evaluation job is deleted, the corresponding result display page will be deleted and cannot be restored. Exercise caution when performing this operation.

### 6.4.6 Model Evaluation API

Custom inference code can calculate inference results and call the **analyse** evaluation API based on the rules defined by ModelArts.

- **API Description (Universal API)**
- **API Description (Custom Evaluation Metrics)**

#### API Description (Universal API)

ModelArts provides the **analyse** API to save the inference result in a specified format. You need to call this API based on the following rules after the inference is complete.

```python
analyse(task_type='',
         pred_list=[],
         label_list=[],
         name_list=[],
         custom_metric='',
         label_map_dict=''
)
```
### Table 6-19 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_type</td>
<td>Yes</td>
<td>Job type. Available job types are <code>image_classification</code> and <code>image_object_detection</code>. <code>image_classification</code> indicates the image classification type. <code>image_object_detection</code> indicates the object detection type.</td>
</tr>
<tr>
<td>pred_list</td>
<td>Yes</td>
<td>List of model prediction outputs.</td>
</tr>
<tr>
<td>label_list</td>
<td>Yes</td>
<td>List of all image labels.</td>
</tr>
<tr>
<td>name_list</td>
<td>Yes</td>
<td>OBS paths of all images. Use absolute paths.</td>
</tr>
<tr>
<td>custom_metric</td>
<td>No</td>
<td>Custom metric.</td>
</tr>
<tr>
<td>label_map_dict</td>
<td>No</td>
<td>Label index and name. If this parameter is not set, the system uses <code>{&quot;0&quot;: &quot;0&quot;, &quot;1&quot;: &quot;1&quot;, &quot;2&quot;: &quot;2&quot;, ...}</code> as the display label by default, for example, <code>{&quot;0&quot;: &quot;dog&quot;, &quot;1&quot;: &quot;cat&quot;, &quot;2&quot;: &quot;horse&quot;}</code>.</td>
</tr>
</tbody>
</table>

**pred_list, label_list, and name_list** must be Python list objects with the same length. The objects in the three lists must be in one-to-one mapping. For example, the first element of **pred_list** is the prediction result of the first image, the first element of **label_list** is the label of the first image, and the first element of **name_list** is the absolute path of the first image.

**name_list** indicates the path for storing images on OBS. The related metrics are sensitivity analysis and inference result viewing, which must be consistent with **pred** and **label_list**. The following are examples:

```
['obs://test/cat/xxx.jpg', ..., 'obs://test/dog/yyy.jpg']
```

The following is an example of **pred_list** in the evaluation code for the model of the image classification type. The elements of **pred_list** are the one-dimensional NumPy ndarray or one-dimensional Python list. The length is the number of classes. **pred_list** indicates the confidence score of an image in each class.

```
[0.87, 0.11, 0.02],
[0.1, 0.7, 0.2],
[0.03, 0.04, 0.93],
[0.25, 0.65, 0.1],
[0.3, 0.34, 0.36]
```

The following is an example of **label_list**. The elements of **label_list** are integers, indicating the label classes of the image.

```
[0, 1, 2, 1, 2]
```

In the evaluation code for image semantic segmentation, the elements of **pred_list** are the inference and classification results of each pixel in the image, and the shape is the same as the image size.
The elements of `label_list` are the label class of each pixel in the image. The shape is the same as the image size.

The following is an example of `pred_list` in the evaluation code for the model of the object detection type. The Python list contains three elements. The first element is a two-dimensional array or NumPy ndarray object, whose shape is `num (number of bounding boxes in an image) x 4(ymin, xmin, ymin, xmax)`. The second element is a one-dimensional array or NumPy ndarray object, whose length is `num (number of bounding boxes in an image)`. The third element is a one-dimensional array or NumPy ndarray object, whose length is `num (number of bounding boxes in an image)`. `pred_list` indicates [target bounding box coordinate, target bounding box class, and confidence score of the class corresponding to the target bounding box].

The following is an example of elements in `label_list`. The Python list contains two elements. The first element is a two-dimensional array or NumPy ndarray object, whose shape is `num (number of bounding boxes in an image) x 4(ymin, xmin, ymin, xmax)`. The second element is a one-dimensional array or NumPy ndarray object, whose length is `num (number of bounding boxes in an image)`. `label_list` indicates [target bounding box coordinate, target bounding box class].
API Description (Custom Evaluation Metrics)

If you want to draw custom evaluation metrics on the GUI, you only need to generate JSON formats based on certain rules and write the formats to the generated JSON files.

The following provides some display modes and corresponding JSON formats. You can combine multiple styles into a complete JSON format. For details, see JSON format consisting of multiple styles.

- **Chart style**
- **Table style**
- **Image style**
  - Chart style

**Figure 6-17** Example of the chart style

JSON format

```json
'line_chart': {'name': 'x_axis_name': str,
  'y_axis_name': str,
  'x_axis_range': [x_min, x_min+x_step, x_min+2*x_step, ..., x_max],
  'y_axis_range': [y_min, y_min+y_step, y_min+2*y_step, ..., y_max],
```
Pie chart: Values or percentages are displayed using the pie chart.
```
'pie_chart': {'name': 'label_1': [(x0, y0), (x1, y1), ...],
              'value': [value_1, value_2, ...]}
```

Column chart: Values are displayed using columns.
```
'column_chart': {'name': 'x_axis_name': str,
                 'y_axis_name': str,
                 'x_axis_range': [x_min, x_min+x_step, x_min+2*x_step, ..., x_max],
                 'y_axis_range': [y_min, y_min+y_step, y_min+2*y_step, ..., y_max],
                 'x_value': [value_1, value_2, ...value_n+1],
                 'y_value': [value_1, value_2, ...value_n]}
```

**Table style**
```
'table': {'name': 'top_left_cell': 'cell text',
          'row_labels': ['name_1', 'name_2', ..., 'name_m'],
          'col_labels': ['name_1', 'name_2', ..., 'name_n'],
          'cell_value': [[v11, v12, v13, ..., v1n], [], ..., [vm1, vm2, ..., vmn]]}
```

**Image style**

Figure 6-18 Example of the image style

JSON format
```
"get_negative_samples_cls": {
  "labels": [
    {
      "name": "Class name",
      "type": 0,
      "property": {
        ...
      }
    },
    ...
  ],
  "predict_labels": [
    {
      "name": "Class name",
      "type": 0,
      "property": {
        ...
      }
    },
    ...
  ],
  "score": "0.424",
  "data_info": "/data/leedsbutterfly/images/0090180.png"
}
```

**JSON format consisting of multiple styles**
You can combine multiple styles into a complete JSON format. The following example shows the JSON format consisting of a line chart and a table. You need to enter the description and title.
6.4.7 Sample Code for Model Evaluation

Sample code is provided for common scenarios such as image classification, image semantic segmentation, and object detection. You can compile your evaluation code based on the sample code.

- **Sample Code for Models of the Image Classification Type**
- **Sample Code for Models of the Image Semantic Segmentation Type**
- **Sample Code for Models of the Object Detection Type**

### Sample Code for Models of the Image Classification Type

The training model corresponding to the following sample code is the built-in algorithm **ResNet_v1_50** (TensorFlow engine).

- **model_url**: model directory. After a model version is selected on the GUI, this parameter is automatically added in the background.
- **data_url**: dataset directory. After a dataset version is selected on the GUI, this parameter is automatically added in the background.

```python
import json
import logging
import os
import sys
import tempfile
import h5py
import numpy as np
from PIL import Image
import moxing as mox
import tensorflow as tf
from deep_moxing.framework.manifest_api.manifest_api import get_sample_list
from deep_moxing.model_analysis.api import analyse, tmp_save
from deep_moxing.model_analysis.common.constant import TMP_FILE_NAME

logging.basicConfig(level=logging.DEBUG)
FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string('model_url', '', 'path to saved model')
```

---

ModelArts
User Guide (Senior AI Engineers)

Issue 01 (2021-01-15) Copyright © Huawei Technologies Co., Ltd. 286
tf.app.flags.DEFINE_string('data_url', '', 'path to output files')
tf.app.flags.DEFINE_string('adv_param_json', '{"attack_method":"i-FGSM","eps":30, "iter_times":4}', 'params for attacks')
FLAGS(sys.argv, known_only=True)

def _preprocess(data_path):
    img = Image.open(data_path)
    img = img.convert('RGB')
    img = np.asarray(img, dtype=np.float32)
    img = img[np.newaxis, :, :, :]
    return img

def softmax(x):
    x = np.array(x)
    orig_shape = x.shape
    if len(x.shape) > 1:
        # Matrix
        x = np.apply_along_axis(lambda x: np.exp(x - np.max(x)), 1, x)
        denominator = np.apply_along_axis(lambda x: 1.0 / np.sum(x), 1, x)
        if len(denominator.shape) == 1:
            denominator = denominator.reshape((denominator.shape[0], 1))
        x = x * denominator
    else:
        # Vector
        x_max = np.max(x)
        x = x - x_max
        numerator = np.exp(x)
        denominator = 1.0 / np.sum(numerator)
        x = numerator.dot(denominator)
    assert x.shape == orig_shape
    return x

def get_dataset(data_path, label_map_dict):
    label_list = []
    img_name_list = []
    if 'manifest' in data_path:
        manifest, _ = get_sample_list(
            manifest_path=data_path,
            task_type='image_classification')
        for item in manifest:
            if len(item[1]) != 0:
                label_list.append(label_map_dict.get(item[1][0]))
                img_name_list.append(item[0])
            else:
                continue
    else:
        label_name_list = os.listdir(data_path)
        label_dict = {}
        for idx, item in enumerate(label_name_list):
            label_dict[str(idx)] = item
            sub_img_list = os.listdir(os.path.join(data_path, item))
            img_name_list += [os.path.join(data_path, item, img_name) for img_name in sub_img_list]
        label_list += [label_map_dict.get(item)] * len(sub_img_list)
    return img_name_list, label_list

def _preprocess(data_path):
    img = Image.open(data_path)
    img = img.convert('RGB')
    img = np.asarray(img, dtype=np.float32)
    img = img[np.newaxis, :, :, :]
    return img

def softmax(x):
    x = np.array(x)
    orig_shape = x.shape
    if len(x.shape) > 1:
        # Matrix
        x = np.apply_along_axis(lambda x: np.exp(x - np.max(x)), 1, x)
        denominator = np.apply_along_axis(lambda x: 1.0 / np.sum(x), 1, x)
        if len(denominator.shape) == 1:
            denominator = denominator.reshape((denominator.shape[0], 1))
        x = x * denominator
    else:
        # Vector
        x_max = np.max(x)
        x = x - x_max
        numerator = np.exp(x)
        denominator = 1.0 / np.sum(numerator)
        x = numerator.dot(denominator)
    assert x.shape == orig_shape
    return x

def get_dataset(data_path, label_map_dict):
    label_list = []
    img_name_list = []
    if 'manifest' in data_path:
        manifest, _ = get_sample_list(
            manifest_path=data_path,
            task_type='image_classification')
        for item in manifest:
            if len(item[1]) != 0:
                label_list.append(label_map_dict.get(item[1][0]))
                img_name_list.append(item[0])
            else:
                continue
    else:
        label_name_list = os.listdir(data_path)
        label_dict = {}
        for idx, item in enumerate(label_name_list):
            label_dict[str(idx)] = item
            sub_img_list = os.listdir(os.path.join(data_path, item))
            img_name_list += [os.path.join(data_path, item, img_name) for img_name in sub_img_list]
        label_list += [label_map_dict.get(item)] * len(sub_img_list)
    return img_name_list, label_list

def deal_ckpt_and_data_with_obs():
    pb_dir = FLAGS.model_url
    data_path = FLAGS.data_url

    if pb_dir.startwith('obs://'):
        mox.file.copy_parallel(pb_dir, '/cache/ckpt/')
    pb_dir = '/cache/ckpt'
    print('------------ download success ----------')
if data_path.startswith('obs:// '):
    mox.file.copy_parallel(data_path, '/cache/data/')
data_path = '/cache/data/
print('------------- download dataset success ------------')
assert os.path.isdir(pb_dir), 'Error, pb_dir must be a directory'
return pb_dir, data_path

def evaluation():
    pb_dir, data_path = deal_ckpt_and_data_with_obs()
    adv_param_json = FLAGS.adv_param_json
    index_file = os.path.join(pb_dir, 'index')
    try:
        label_file = h5py.File(index_file, 'r')
        label_array = label_file['labels_list'][:].tolist()
        label_array = [item.decode('utf-8') for item in label_array]
    except Exception as e:
        logging.warning(e)
        logging.warning('index file is not a h5 file, try json.')
        with open(index_file, 'r') as load_f:
            label_file = json.load(load_f)
        label_array = label_file['labels_list'][:]
    label_map_dict = {}
    label_dict = {}
    for idx, item in enumerate(label_array):
        label_map_dict[item] = idx
        label_dict[idx] = item
    print(label_map_dict)
    print(label_dict)
    data_file_list, label_list = get_dataset(data_path, label_map_dict)
    assert len(label_list) > 0, 'missing valid data'
    assert None not in label_list, 'dataset and model not match'
pred_list = []
file_name_list = []
for img_path in data_file_list:
    img = _preprocess(img_path)
    img_list.append(img)
    file_name_list.append(img_path)
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
with tf.Session(graph=tf.Graph(), config=config) as sess:
    meta_graph_def = tf.saved_model.loader.load(
        sess, [tf.saved_model.tag_constants.SERVING], pb_dir)
    signature = meta_graph_def.signature_def
    signature_key = 'predict_object'
    input_key = 'images'
    output_key = 'logits'
x_tensor_name = signature[signature_key].inputs[input_key].name
y_tensor_name = signature[signature_key].outputs[output_key].name
x = sess.graph.get_tensor_by_name(x_tensor_name)
y = sess.graph.get_tensor_by_name(y_tensor_name)
for img in img_list:
    pred_output = sess.run([y], {x: img})
pred_output = softmax(pred_output[0])
pred_list.append(pred_output[0].tolist())
label_dict = json.dumps(label_dict)
task_type = 'image_classification'

# analyse
res = analyse(
    task_type=task_type,
    pred_list=pred_list,
    label_list=label_list,
Sample Code for Models of the Image Semantic Segmentation Type

The following sample code corresponds to the D-LinkNet road segmentation model (using the TensorFlow engine).

- **model_url**: model directory. After a model version is selected on the GUI, this parameter is automatically added in the background.
- **data_url**: dataset directory. After a dataset version is selected on the GUI, this parameter is automatically added in the background.

```python
import glob
import json
import logging
import os
import sys
import numpy as np
from PIL import Image
import moxing as mox
import tensorflow as tf
from deep_moxing.model_analysis.api import analyse
logging.basicConfig(level=logging.DEBUG)
FLAGS = tf.app.flags.FLAGS

# Define model parameters

def _norm(img):
    mean = np.mean(img, axis=(0, 1), keepdims=True)
    std = np.std(img, axis=(0, 1), keepdims=True)
    img = (img - mean) / std
    return img

def _preprocess(data_path):
    img = Image.open(data_path)
    img = img.convert('RGB')
    img = np.asarray(img, dtype=np.float32)
    img = _norm(img)
    img = img[np.newaxis, :, :, :]
    return img

def evalution():
    pb_dir = FLAGS.model_url
    data_path = FLAGS.data_url
    if data_path.startswith('obs://'):
        mox.file.copy_parallel(data_path, '/cache/dataset')
    else:
        image_data_path = os.path.join(data_path, 'eval_uint8')
        label_path = os.path.join(data_path, 'eval_label')
    if pb_dir.startswith('obs://'):
        mox.file.copy_parallel(pb_dir, '/cache/model')
```
Sample Code for Models of the Object Detection Type

The training model corresponding to the following example code is the built-in algorithm **Faster_RCNN_ResNet_v1_50** (TensorFlow engine).

- **model_url**: model directory. After a model version is selected on the GUI, this parameter is automatically added in the background.
• **data_url**: dataset directory. After a dataset version is selected on the GUI, this parameter is automatically added in the background.

```python
import moxing as mox
from deep_moxing.model_analysis.api import analyse
from deep_moxing.framework.manifest_api.manifest_api import get_list
import tensorflow as tf
import Image
import numpy as np
import xml.etree.ElementTree as ET
import h5py
import os
import json
import logging
import time
import sys
logging.basicConfig(level=logging.DEBUG)
FLAGS = tf.app.flags.FLAGS
FLAGS(sys.argv, known_only=True)
def _get_label(label_path, label_map_dict):
    root = ET.parse(label_path).getroot()
    bbox_list = []
    label_list = []
    for obj in root.iter('object'):
        xml_box = obj.find('bndbox')
        xmin = int(float(xml_box.find('xmin').text))
        ymin = int(float(xml_box.find('ymin').text))
        xmax = int(float(xml_box.find('xmax').text))
        ymax = int(float(xml_box.find('ymax').text))
        label_name = obj.find('name').text
        bbox_list.append([ymin, xmin, ymax, xmax])
        label_list.append(label_map_dict.get(label_name))
    assert None not in label_list, 'dataset and model not match'
    return [bbox_list, label_list]
def _preprocess(data_path):
    img = Image.open(data_path)
    img = img.convert('RGB')
    img = np.asarray(img, dtype=np.float32)
    img = img[np.newaxis, :, :, :]
    return img
def get_data_ckpt_local():
    pb_dir = FLAGS.model_url
    data_path = FLAGS.data_url
    data_file_list = []
    label_file_list = []
    if 'manifest' in data_path:
        data_file_list, label_file_list = get_list(manifest_path=data_path)
        print('-------------- download ------------
        mox.file.copy_parallel(pb_dir, '/cache/ckpt/
        pb_dir = '/cache/ckpt/
        print('-------------- download success ------------
        elif data_path.startswith('obs://'):
        print('-------------- download ------------
        mox.file.copy_parallel(pb_dir, '/cache/ckpt/
        mox.file.copy_parallel(data_path, '/cache/data/
        pb_dir = '/cache/ckpt/
        data_path = '/cache/data/
        print('-------------- download success ------------
    if pb_dir:
```
assert os.path.isdir(pb_dir), 'Error, pb_dir must be a directory'

index_file = os.path.join(pb_dir, 'index')
label_list = []
file_name_list = []
img_list = []
try:
    label_file = h5py.File(index_file, 'r')
    label_array = label_file['labels_list'][:].tolist()
    label_array = [item.decode('utf-8') for item in label_array]
except Exception as e:
    logging.warning(e)
    logging.warning('index file is not a h5 file, try json.')
    with open(index_file, 'r') as load_f:
        label_file = json.load(load_f)
    label_array = label_file['labels_list'][:]
label_map_dict = {}
label_dict = {}
for idx, item in enumerate(label_array):
    label_map_dict[item] = idx
    label_dict[idx] = item
if 'manifest' in data_path:
    for img_path, xml_path in zip(data_file_list, label_file_list):
        label = _get_label(xml_path, label_map_dict)
        img = _preprocess(img_path)
        label_list.append(label)
        img_list.append(img)
        file_name_list.append(img_path)
else:
    file_list = os.listdir(data_path)
    for item in file_list:
        if ('jpg' in item) or ('bmp' in item) or ('png' in item):
            xml_path = os.path.join(data_path, item.split('.')[0] + '.xml')
            img_path = os.path.join(data_path, item)
            label = _get_label(xml_path, label_map_dict)
            img = _preprocess(img_path)
            label_list.append(label)
            img_list.append(img)
            file_name_list.append(img_path)
        else:
            continue
assert len(label_list) > 0, 'missing valid data'
return pb_dir, label_list, label_dict, file_name_list, img_list

def evaluation():
    pred_list = []
    pb_dir, label_list, label_dict, file_name_list, img_list = get_data_ckpt_local()
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    config.gpu_options.visible_device_list = '0'
    with tf.Session(graph=tf.Graph(), config=config) as sess:
        meta_graph_def = tf.saved_model.loader.load(sess, [tf.saved_model.tag_constants.SERVING], pb_dir)
        signature = meta_graph_def.signature_def
        signature_key = 'predict_object'
        input_key = 'images'
        output_key0 = 'detection_boxes'
        output_key1 = 'detection_classes'
        output_key2 = 'detection_scores'
        x_tensor_name = signature[signature_key].inputs[input_key].name
        y_tensor_name0 = signature[signature_key].outputs[output_key0].name
        y_tensor_name1 = signature[signature_key].outputs[output_key1].name
        y_tensor_name2 = signature[signature_key].outputs[output_key2].name
        x = sess.graph.get_tensor_by_name(x_tensor_name)
        y0 = sess.graph.get_tensor_by_name(y_tensor_name0)
        y1 = sess.graph.get_tensor_by_name(y_tensor_name1)
        y2 = sess.graph.get_tensor_by_name(y_tensor_name2)
        start = time.time()
for img in img_list:
    pred_detection_boxes, pred_detection_classes, \
    pred_detection_scores = sess.run([y0, y1, y2], {x: img})
    if pred_detection_boxes.ndim == 3:
        pred_detection_boxes = pred_detection_boxes[0]
        pred_detection_classes = pred_detection_classes[0]
        pred_detection_scores = pred_detection_scores[0]
        pred_list.append([\n            pred_detection_boxes.tolist(),
            (pred_detection_classes - 1).tolist(),
            pred_detection_scores.tolist()\n        ])
end = time.time()
fps = len(img_list) / (end - start)
diy_metric = {'fps': {'value': {'fps': fps}}}
label_dict = json.dumps(label_dict)
task_type = 'image_object_detection'
# analyse
res = analyse(
    task_type=task_type,
    pred_list=pred_list,
    label_list=label_list,
    name_list=file_name_list,
    custom_metric=diy_metric,
    label_map_dict=label_dict)

if __name__ == '__main__':
evalution()

6.4.8 Model Optimization Suggestions

6.4.8.1 Analysis on Class Errors of Object Detection Models and Solution

Symptom

During the model evaluation phase of object detection, false positive analysis and
false negative analysis are important.

- In false positive analysis, the class error is mainly reflected by the probability
  of class false positive.
- In false negative analysis, the class error is mainly reflected by the probability
  of class false negative.

The model evaluation results in ModelArts are as follows:

**Figure 6-19 Model evaluation results**

The preceding figure shows an example of analysis on the object detection
inference results. Graph a shows the analysis on the position false positive, class
false positive, and background false positive. Graph b shows the analysis on the position false negative, class false negative, and background false negative.

In the preceding figure, class errors account for a large proportion of false positive and false negative errors. You are advised to refer to the following algorithms and technical description to understand how to reduce the class error probability and improve model inference accuracy.

**Solution**

Multiscale is a common data processing mode in object detection. You can define the scales in your models. During model training, you are advised to perform scaling at a fixed interval (steps) to ensure the multiscale input of the images in the training of a single epoch. The multiscale training process is as follows:

**Figure 6-20** Multiscale training process

![Multiscale training process](image)

**Verification**

A dataset for camera type identification is used for verification. The dataset has 14 classes, which represent 14 different cameras. The comparison between the false positive analysis before and after multiscale is used shows that multiscale significantly reduces the class false positive ratio.

**Figure 6-21** Analysis and comparison of false positives after multiscale is used

(a) False positive analysis before multiscale is used

(b) False positive analysis after multiscale is used

The comparison between the false negative analysis before and after multiscale is used shows that multiscale significantly reduces the class false negative ratio.
Figure 6-22 Analysis and comparison of false negatives after multiscale is used

(a) False negative analysis before multiscale is used

(b) False negative analysis after multiscale is used

Suggestions

In the model inference result, if the class error ratio is large, you are advised to use multiscale to optimize model training.

6.4.8.2 Analysis on Position Errors of Object Detection Models and Solution

Symptom

During the model evaluation phase of object detection, false positive analysis and false negative analysis are important. In false positive analysis, the position error is mainly reflected by the probability of position false positive. In false negative analysis, the position error is mainly reflected by the probability of position false negative.

The model evaluation results in ModelArts are as follows:

Figure 6-23 Model evaluation results

(a) False positive analysis

(b) False negative analysis

The preceding figure shows an example of analysis on the object detection inference results. Graph a shows the analysis on the position false positive, class false positive, and background false positive. Graph b shows the analysis on the position false negative, class false negative, and background false negative.

In the preceding figure, position errors account for a large proportion of false positive and false negative errors. You are advised to refer to the following algorithms and technical description to understand how to reduce the position error probability and improve model inference accuracy.

Solution

- **GIoU loss**

  The generalized IoU (GIoU) loss was first proposed by Stanford University. It overcomes the following disadvantages of the IoU loss:
a. When predicted bounding box A and target bounding box B do not intersect, the IoU(A,B) is 0, which cannot accurately reflect the location and distance relationships between the two bounding boxes. In this case, the loss function cannot be derived. The IoU loss cannot be used for inference optimization where two bounding boxes do not intersect.

b. Assume that the sizes of the predicted bounding box and the target bounding box are fixed. If the intersection area size of the two boxes is fixed (that is, the IoUs are the same), the IoUs cannot reflect how the two boxes intersect. The following figure shows three scenarios where the predicted bounding box and the target bounding box intersect. In the three scenarios, the IoUs are the same but the position relationships are greatly different.

**Figure 6-24** Position relationships between the predicted bounding box and the target bounding box when the IoUs are the same

The following figure shows the formula for calculating the GIoU. In this formula, A indicates the predicted bounding box, B indicates the target bounding box, and C indicates the intersection area between A and B.

**Figure 6-25** GIoU calculation

\[
GIoU = IoU - \frac{|C \setminus (A \cup B)|}{|C|}
\]

Formula for calculating the value of the GIoU loss:

\[
\mathcal{L}_{GIoU} = 1 - GIoU
\]
- **DIOU loss**

  Distance IoU (DIOU) incorporates the distance between the target bounding box and the anchor bounding box, overlapping rate, and scale. In this way, the target bounding box regression becomes more stable during object detection model training. It has the following advantages:

  a. A more direct regression minimizes a normalized distance between the anchor bounding box and the target bounding box, leading to much faster convergence.

  b. If bounding box regression is unavailable using the GIoU loss, for example, the target bounding box contains the predicted bounding box (see the following figure), the bounding boxes can be quickly regressed.

**Figure 6-26** The green bounding box indicates the target bounding box, and the red bounding box indicates the predicted bounding box.

Formula for calculating DIOU:

\[
DIOU = IoU - \frac{\rho^2(b, b^{gt})}{c^2}.
\]

The letters in the formula are described as follows:

The green bounding box indicates the predicted bounding box, and the black bounding box indicates the target bounding box. d indicates the Euclidean distance between the center points of the two boxes, and c indicates the length of the diagonal of the minimum intersection area.

Formula for calculating the DIOU loss:

\[
\mathcal{L}_{DIOU} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2}
\]
Verification

A dataset of traffic signal lights is used for verification. The dataset is used to detect traffic signal lights at the crossroads. There is only one class. The comparison between the false positive analysis before and after the DIoU loss is used shows that the DIoU loss significantly reduces the position false positive ratio.

**Figure 6-27** Analysis and comparison of false positives after the DIoU loss is used

(a) False positive analysis before the DIoU loss is used
(b) False positive analysis after the DIoU loss is used

The comparison between the false negative analysis before and after the DIoU loss is used shows that the DIoU loss significantly reduces the position false negative ratio.

**Figure 6-28** Analysis and comparison of false negatives after the DIoU loss is used

(a) False negative analysis before the DIoU loss is used
(b) False negative analysis after the DIoU loss is used

Suggestions

In the model inference result, if the position error ratio is large, you are advised to use the GIoU loss and DIoU loss for model optimization and enhancement during training.

6.4.8.3 Analysis on Background Errors of Object Detection Models and Solution

Symptom

During the model evaluation phase of object detection, false positive analysis and false negative analysis are important. In false positive analysis, the background error is mainly reflected by the probability of background false positive. In false negative analysis, the background error is mainly reflected by the probability of background false negative.

The model evaluation results in ModelArts are as follows:
The preceding figure shows an example of analysis on the object detection inference results. Graph a shows the analysis on the position false positive, class false positive, and background false positive. Graph b shows the analysis on the position false negative, class false negative, and background false negative.

In the preceding figure, background errors account for a large proportion of false positive and false negative errors. You are advised to refer to the following algorithms and technical description to understand how to reduce the background error probability and improve model inference accuracy.

**Solution**

Object detection involves multi-task training. In addition to accurate class identification, class instances need to be accurately located. The model training loss includes the class loss and bounding box loss. The most common bounding box loss is the Smooth L1 loss. The calculation formula is as follows:
The balanced loss was first proposed in Libra R-CNN. Compared with the traditional Smooth L1 loss, the balanced loss has a more smooth curve and better convergence. The specific calculation formula and derivation formula are as follows:

\[
L_{1\text{smooth}} = \begin{cases} 
0.5x^2 & |x| < 1 \text{(inliers)} \\
|x| - 0.5 & \text{otherwise (outliers)}
\end{cases}
\]

\[
\frac{\partial L_{1\text{smooth}}}{\partial x} = \begin{cases} 
x & |x| < 1 \\
-1 & x < -1 \\
1 & x > 1
\end{cases}
\]

The balanced loss was first proposed in Libra R-CNN. Compared with the traditional Smooth L1 loss, the balanced loss has a more smooth curve and better convergence. The specific calculation formula and derivation formula are as follows:

\[
L_b(x) = \begin{cases} 
\frac{\alpha}{b}(b|x| + 1)\ln(b|x| + 1) - \alpha|x| & \text{if } |x| < 1 \\
\gamma|x| + C & \text{otherwise}
\end{cases}
\]

\[
\frac{\partial L_b}{\partial x} = \begin{cases} 
\alpha \ln(b|x| + 1) & \text{if } |x| < 1 \\
\gamma & \text{otherwise}
\end{cases}
\]

Compared with the Smooth L1 loss, the balanced loss has a slightly larger gradient at the boundary of inliers. A smaller Alpha value means more obvious increase. In this way, the balanced loss can increase the probability of identifying positive samples during the update of the model reverse gradient.
Verification

The open source dataset Canine Coccidiosis Parasite is used for verification. The following figure shows the comparison of false positives before and after the balanced loss is used in the dataset. The balanced loss significantly reduces the background false positive ratio.

**Figure 6-30** Analysis and comparison of false positives before and after the balanced loss is used

The following figure shows the comparison of false negatives before and after the balanced loss is used in Canine Coccidiosis Parasite. The balanced loss significantly reduces the background false negative ratio.

**Figure 6-31** Analysis and comparison of false negatives before and after the balanced loss is used

**Suggestions**

In the model inference result, if the background error ratio is large, you are advised to use the balanced loss for model optimization and enhancement during training.
6.4.8.4 Analysis on the Sensitivity of Object Detection Models to Bounding Box Aspect Ratios and Solution

Symptom

In an object detection task, different bounding boxes in an image have various shapes, which can be reflected by the aspect ratios of bounding boxes. A wider range of bounding box aspect ratios means more unbalanced distribution of bounding box shapes in the database. Object detection models have different effects on datasets with different aspect ratios. The following describes how to reduce the sensitivity of models to the aspect ratios of bounding boxes.

In the following figure, there are three aspect ratios of the bounding boxes in the image.

Figure 6-32 Example of aspect ratios of bounding boxes

Solution

In an object detection task, the Feature Pyramid Networks (FPN) is widely used in one-stage detection models. The FPN performs concatenation on feature maps with different scales through feature fusion, and then performs regression on subsequent classes and bounding boxes. This has become a standard method for object detection models. The EfficientDet thesis puts forwards an FPN block repeats method. In this method, FPN is regarded as a basic unit and repeats for feature extraction and fusion. The following figure shows the Bidirectional Feature Pyramid Network (BiFPN), a type of FPN.
The following figure shows the FPN block repeats method. The feature extraction layer is deeper.

**Figure 6-34** FPN block repeats of EfficientDet

The FPN block repeats technology is not only applicable to BiFPN, but also to other FPN structures, such as PANet FPN. The following figure shows that the FPN block repeats technology is applied to PANet.

In this figure, graph a shows a basic structure of PANet, and graph b shows a network with FPN block repeats.
**Verification**

The open source dataset fruit is used for verification. Before using FPN block repeats, analyze the aspect ratio sensitivity of bounding boxes. The following table indicates that the aspect ratio sensitivity of Apple is 0.0757 and that of Banana is 0.4481.

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>Apple</th>
<th>Banana</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>1</td>
<td>0.5714</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>0.875</td>
<td>0</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>0.8182</td>
<td>1</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>0.8571</td>
<td>0</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.0757</td>
<td>0.4481</td>
</tr>
</tbody>
</table>

After the FPN block repeats technology is used, the aspect ratio sensitivity of bounding boxes is analyzed. As shown in the following figure, the aspect ratio sensitivity of the Apple bounding boxes decreases from 0.0757 to 0.0667, and that of the Banana bounding boxes decreases from 0.4481 to 0.4091.

The FPN block repeats technology significantly improves the aspect ratio sensitivity of bounding boxes.

**Table 6-21** Analysis on the sensitivity of object detection models to bounding box aspect ratios after FPN block repeats is used

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>Apple</th>
<th>Banana</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>1</td>
<td>0.7857</td>
</tr>
</tbody>
</table>
### Feature Distribution

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>Apple</th>
<th>Banana</th>
</tr>
</thead>
<tbody>
<tr>
<td>20% - 40%</td>
<td>0.8333</td>
<td>0</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.0667</td>
<td>0.4091</td>
</tr>
</tbody>
</table>

### Suggestions

In the model inference result, if the detected class is very sensitive to the aspect ratio of bounding boxes, you are advised to use the object detection models with FPN block repeats for optimization during training.

### 6.4.8.5 Analysis on the Sensitivity of Object Detection Models to Bounding Box Brightness and Solution

#### Symptom

In an object detection task, the bounding box brightness of different datasets may be different, which can be reflected by bounding box brightness sensitivity. Bounding box brightness affects model training and inference. The following figure shows brightness values of the labels in the bounding boxes in an image.

**Figure 6-36** Uneven brightness of bounding boxes

#### Solution

Dropout is a widely used regularization technology in deep learning tasks. However, this technology randomly drops a unit at the feature layer. As a result, semantic information shared by adjacent feature units is also dropped. DropBlock solves the preceding problem. It drops features in a feature block (a contiguous...
region of a feature map), and performs regularization on the deep learning network. DropBlock is a simple method similar to Dropout. The main difference between the two is that DropBlock drops the adjacent regions of a feature map instead of a separate random unit. For details, see the DropBlock thesis.

DropBlock has two parameters: **block_size** and **γ**.

- **block_size**: block size (length and width). When **block_size** is 1, DropBlock degrades to the traditional Dropout. Available values are 3, 5, and 7.
- **γ**: probability during the drop process, that is, the Bernoulli probability.

The following shows the comparison between Dropout and DropBlock. In the figure, graph b indicates Dropout, and graph c indicates DropBlock.

**Figure 6-37 Principle comparison between Dropout and DropBlock**

---

Official implementation of TensorFlow:

class Dropblock(object):
    """DropBlock: a regularization method for convolutional neural networks. DropBlock is a form of structured dropout, where units in a contiguous region of a feature map are dropped together. DropBlock works better than dropout on convolutional layers due to the fact that activation units in convolutional layers are spatially correlated. See https://arxiv.org/pdf/1810.12890.pdf for details."
    """"
    def __init__(self,
        dropblock_keep_prob=None,
        dropblock_size=None,
        data_format='channels_last'):
        self._dropblock_keep_prob = dropblock_keep_prob
        self._dropblock_size = dropblock_size
        self._data_format = data_format

    def __call__(self, net, is_training=False):
        """Builds Dropblock layer.
        Args:
            net: 'Tensor' input tensor.
            is_training: 'bool' if True, the model is in training mode.
        Returns:
            A version of input tensor with DropBlock applied.
        """"
        if (not is_training or self._dropblock_keep_prob is None or
            self._dropblock_keep_prob == 1.0):
            return net

        logging.info('Applying DropBlock: dropblock_size %d,' % self._dropblock_size,
                     'net.shape %s', self._dropblock_size, net.shape)

        if self._data_format == 'channels_last':
The open source dataset Canine Coccidiosis Parasite is used for verification. The dataset has only one class. Before DropBlock is used, Table 6-22 describes the sensitivity of models to bounding box brightness before DropBlock is used.

### Table 6-22 Brightness sensitivity analysis of bounding boxes

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>coccidia</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>0.8065</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>0.871</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>0.9355</td>
</tr>
</tbody>
</table>
Table 6-23 describes the sensitivity of models to bounding box brightness before DropBlock is used. After DropBlock is used, the brightness sensitivity of the bounding boxes reduces from 0.0658 to 0.0204. DropBlock can significantly reduces the bounding box brightness sensitivity of the object detection models in model evaluation.

### Table 6-23 Brightness sensitivity analysis of bounding boxes

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>coccidia</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>0.9355</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>0.9677</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>0.9677</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>0.9677</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>1</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.0204</td>
</tr>
</tbody>
</table>

**Suggestions**

In the model inference result, if the detected classes are very sensitive to the brightness of bounding boxes, you are advised to use DropBlock for model optimization and enhancement during training.

### 6.4.8.6 Analysis on the Sensitivity of Object Detection Models to Bounding Box Clarity and Solution

**Symptom**

In an object detection task, the bounding box clarity of different datasets may be different, which can be reflected by bounding box clarity sensitivity. Bounding box clarity affects model training and inference.

The left graph shows the original image, and the right graph shows that the clarity of a bounding box changes.
Solution

Dropout is a widely used regularization technology in deep learning tasks. However, this technology randomly drops a unit at the feature layer. As a result, semantic information shared by adjacent feature units is also dropped. DropBlock solves the preceding problem. It drops features in a feature block (a contiguous region of a feature map), and performs regularization on the deep learning network. DropBlock is a simple method similar to Dropout. The main difference between the two is that DropBlock drops the adjacent regions of a feature map instead of a separate random unit. For details, see the DropBlock thesis.

DropBlock has two parameters: block_size and γ.

- **block_size**: block size (length and width). When block_size is 1, DropBlock degrades to the traditional Dropout. Available values are 3, 5, and 7.
- **γ**: probability during the drop process, that is, the Bernoulli probability.

The following shows the comparison between Dropout and DropBlock. In the figure, graph b indicates Dropout, and graph c indicates DropBlock.

**Figure 6-39 Principle comparison between Dropout and DropBlock**
class Dropblock(object):
    """DropBlock: a regularization method for convolutional neural networks. DropBlock is a form of structured dropout, where units in a contiguous region of a feature map are dropped together. DropBlock works better than dropout on convolutional layers due to the fact that activation units in convolutional layers are spatially correlated. See https://arxiv.org/pdf/1810.12890.pdf for details."
    def __init__(self, dropblock_keep_prob=None, dropblock_size=None, data_format='channels_last'):
        self._dropblock_keep_prob = dropblock_keep_prob
        self._dropblock_size = dropblock_size
        self._data_format = data_format
    def __call__(self, net, is_training=False):
        """Builds Dropblock layer."
        Args:
            net: `Tensor` input tensor.
            is_training: `bool` if True, the model is in training mode.
        Returns:
            A version of input tensor with DropBlock applied.
        """
        if (not is_training or self._dropblock_keep_prob is None or self._dropblock_keep_prob == 1.0):
            return net
        logging.info('Applying DropBlock: dropblock_size %d, net.shape %s', self._dropblock_size, net.shape)
        if self._data_format == 'channels_last':
            _, height, width, _ = net.get_shape().as_list()
        else:
            _, _, height, width = net.get_shape().as_list()
        total_size = width * height
        dropblock_size = min(self._dropblock_size, min(width, height))
        # Seed_drop_rate is the gamma parameter of DropBlcok.
        seed_drop_rate = (1.0 - self._dropblock_keep_prob) * total_size / dropblock_size**2 / ((width - self._dropblock_size + 1) * (height - self._dropblock_size + 1))
        # Forces the block to be inside the feature map.
        w_i, h_i = tf.meshgrid(tf.range(width), tf.range(height))
        valid_block = tf.logical_and(tf.logical_and(w_i >= int(dropblock_size // 2), w_i < width - (dropblock_size - 1) // 2), tf.logical_and(h_i >= int(dropblock_size // 2), h_i < height - (dropblock_size - 1) // 2))
        if self._data_format == 'channels_last':
            valid_block = tf.reshape(valid_block, [1, height, width, 1])
        else:
            valid_block = tf.reshape(valid_block, [1, 1, height, width])
        randnoise = tf.random_uniform(net.shape, dtype=tf.float32)
        valid_block = tf.cast(valid_block, dtype=tf.float32)
        seed_keep_rate = tf.cast(1 - seed_drop_rate, dtype=tf.float32)
        block_pattern = (1 - valid_block + seed_keep_rate + randnoise) >= 1
        block_pattern = tf.cast(block_pattern, dtype=tf.float32)
        if self._data_format == 'channels_last':
            ksize = [1, self._dropblock_size, self._dropblock_size, 1]
        else:
            ksize = [1, 1, self._dropblock_size, self._dropblock_size]
        block_pattern = tf.nn.max_pool
Verification

The open source dataset Canine Coccidiosis Parasite is used for verification. The dataset has only one class. Before DropBlock is used, Table 6-24 describes the sensitivity of models to bounding box clarity before DropBlock is used.

Table 6-24 Clarity sensitivity analysis of bounding boxes

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>coccidia</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>0.9355</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>0.9355</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>0.9355</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>0.7742</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>0.8065</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.0718</td>
</tr>
</tbody>
</table>

After DropBlock is used, the clarity sensitivity of the bounding boxes is analyzed. In Table 6-25, the clarity sensitivity of the bounding boxes decreases from 0.0718 to 0.0204.

The DropBlock technology significantly improves the clarity sensitivity of bounding boxes.

Table 6-25 Clarity sensitivity analysis of bounding boxes

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>coccidia</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>1</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>0.9677</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>0.9677</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>0.9677</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>0.9355</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.0204</td>
</tr>
</tbody>
</table>
Suggestions

In the model inference result, if the detected classes are very sensitive to the clarity of the bounding boxes, you are advised to use DropBlock for model optimization and enhancement during training.

6.4.8.7 Analysis on the Sensitivity of Object Detection Models to Bounding Box Areas and Solution

Symptom

In an object detection task, different bounding boxes in an image have different sizes. Some datasets contain a large number of small objects, whereas some other datasets contain a large number of large objects, which can be reflected by the sensitivity to bounding box areas. An area ratio indicates the ratio of the area of a bounding box to the area of the image. A larger area ratio indicates a larger ratio of the object to the image. The following figure shows the large area ratio of small objects to an image.

Figure 6-40 Large area ratio of small objects to an image

Object detection models have different detection effects for datasets with different area ratios. You are advised to refer to the following algorithms and technical description to understand how to reduce the sensitivity of object detection models to bounding box areas.

Solution

Object detection involves multi-task training. In addition to accurate class identification, class instances need to be accurately located. The model training
loss includes the class loss and bounding box loss. The most common bounding box loss is the Smooth L1 loss. The calculation formula is as follows:

**Figure 6-41 Smooth L1 loss curve**

\[
L_{1\text{smooth}} = \begin{cases} 
0.5x^2 & |x| < 1 (inliers) \\
|x| - 0.5 & \text{otherwise (outliers)} 
\end{cases}
\]

\[
\frac{\partial L_{1\text{smooth}}}{\partial x} = \begin{cases} 
x & |x| < 1 \\
-1 & x < -1 \\
1 & x > 1 
\end{cases}
\]

The balanced loss was first proposed in **Libra R-CNN**. Compared with the traditional Smooth L1 loss, the balanced loss has a more smooth curve and better convergence. The specific calculation formula and derivation formula are as follows:
Compared with the Smooth L1 loss, the balanced loss has a slightly larger gradient at the boundary of inliers. A smaller Alpha value means more obvious increase. In this way, the balanced loss can increase the probability of identifying positive samples during the update of the model reverse gradient.

Graph (a) shows the reciprocal relationship, and graph (b) shows the loss relationship (from Libra R-CNN).

**Figure 6-42** Comparison between the Balanced L1 loss and the Smooth L1 loss

### Verification

The open source dataset Canine Coccidiosis Parasite is used for verification. The dataset has only one class. Before the balanced loss is used, **Table 6-26** describes the sensitivity of a model to the bounding box area after the model is trained and evaluated.

**Table 6-26** Sensitivity assessment of a model to the bounding box area before the Balanced L1 loss is used

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>coccidia</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>0.5806</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>0.871</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>0.9677</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>0.9677</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>1</td>
</tr>
</tbody>
</table>
After the balanced loss is used, Table 6-27 describes the sensitivity of a model to the bounding box area after the model is trained and evaluated.

Table 6-27: Sensitivity assessment of a model to the bounding box area after the Balanced L1 loss is used

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>coccidia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard deviation</td>
<td>0.1546</td>
</tr>
</tbody>
</table>

The balanced loss reduces the area sensitivity from 0.1546 to 0.0912.

**Suggestions**

In the model inference result, if the detected classes are very sensitive to the areas of bounding boxes, you are advised to use the balanced loss for model optimization and enhancement during training.

**6.4.8.8 Analysis on the Sensitivity of Object Detection Models to Bounding Box Marginalization Degrees and Solution**

**Symptom**

In an object detection task, the positions of bounding boxes may be different on an image. Some bounding boxes may reside in the middle of the image, and some other bounding boxes may reside at the edge of the image, which can be reflected by the marginalization degree. The marginalization degree means the ratio of the distance between the center point of a bounding box and the center point of the image to the distance between the center point of the image to the image edge. A larger value indicates that the object is closer to the edge. The following figure shows the scenario where bounding boxes are far away from the center. That is, the bounding boxes are close to the edge.
Object detection models have different detection effects for datasets with different marginalization degrees. You are advised to refer to the following algorithms and technical description to understand how to reduce the sensitivity of object detection models to bounding box marginalization degrees.

**Solution**

- **Box loss weight**
  
  In object detection, the loss values of the class and bounding box coordinates are not balanced. If you simply add the values, small bounding box loss values (for example, small objects) are likely to be ignored during reverse gradient calculation, affecting convergence. Weighting the bounding box loss is a technology used to alleviate the imbalance. That is, the box loss weight is automatically adjusted based on the datasets. The following figure shows the unbalanced class loss and box loss values during object detection model training. The difference between the values is about 100 times.
The following is the pseudo code of TensorFlow for the box loss weight.

def model_fn(inputs, mode):
    ...
    # params is the detection algorithm hyperparameter configuration. Change the value of
    # box_loss_weight.
    total_loss = cls_loss + params['box_loss_weight'] * box_loss

- **Label smoothing**

Label smoothing was first proposed in InceptionV2 and is widely used in classification tasks. If some labels are incorrect or inaccurate, the network may trust the labels and make mistakes. To improve network generalization and avoid this error, when the loss is calculated for the label encoded by one hot, the actual class position is multiplied by a coefficient \((1 - e)\). \(e\) is very small, for example, 0.05, and is sent at a probability of 0.95. The non-labeled class is changed from 0 to 0.05 \((e)\) for loss calculation.

The following figure shows the one-hot code of the original label in an object detection model.

**Figure 6-45 One-hot code of the original label**

The following figure shows the label after label smoothing.

**Figure 6-46 Code of the label after label smoothing**

The following is the pseudo code of the TensorFlow version after label smoothing.

```python
positive_label_mask = tf.equal(targets, 1.0)
if label_smoothing > 0:
    from tensorflow.python.ops import math_ops
    smooth_positives = 1.0 - label_smoothing
    smooth_negatives = label_smoothing / num_classes
    labels = labels * smooth_positives + smooth_negatives
    cross_entropy = (
        tf.nn.sigmoid_cross_entropy_with_logits(labels=labels, logits=logits))
```
Verification

A dataset of traffic signal lights is used for verification. The dataset has only one label. It is used to check whether traffic signal lights are installed at intersections. The following figure shows the class loss curve before and after label smoothing is used. The blue curve indicates the fitting curve before label smoothing is used. As the number of iteration steps increases, overfitting occurs. After label smoothing (gray curve) is used, overfitting can be effectively reduced.

Figure 6-47 Fitting curve of the class loss

![Figure 6-47 Fitting curve of the class loss](image)

The trained object detection model is evaluated, and recall values in different inference cases corresponding to different marginalized feature distributions are obtained. Table 6-28 describes the sensitivity of bounding boxes to marginalization degrees before label smoothing and box loss weight are used.

Table 6-28 Sensitivity of bounding boxes to marginalization degrees before label smoothing and box loss weight are used

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>light</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>0.9412</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>0.8235</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>0.7778</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>1</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>0.8333</td>
</tr>
</tbody>
</table>

Standard deviation 0.0823

Table 6-29 describes the sensitivity of bounding boxes to marginalization degrees after label smoothing and box loss weight are used.

After label smoothing and box loss weight are used, the sensitivity of object detection models to marginalization degrees of bounding boxes reduces from 0.0823 to 0.0448.
### Table 6-29 Sensitivity of bounding boxes to marginalization degrees after label smoothing and box loss weight are used

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>light</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>1</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>0.9412</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>0.8889</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>1</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>1</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.0448</td>
</tr>
</tbody>
</table>

### Suggestions

In the model inference result, if the detected classes are very sensitive to the marginalization degrees of bounding boxes, you are advised to use label smoothing and box loss weight for model optimization and enhancement during training.

### 6.4.8.9 Analysis on the Sensitivity of Object Detection Models to Bounding Box Overlap Degrees and Solution

#### Symptom

In an object detection task, a bounding box of an image may be overlapped by another bounding box, which can be reflected by the overlap degree of bounding boxes. A larger overlap degree indicates that more parts of a bounding box are overlapped by other bounding boxes. Object detection models have different detection effects for datasets with different overlap degrees. The following figure shows the scenario where many bounding boxes are overlapping. You are advised to refer to the following algorithms and technical description to understand how to reduce the sensitivity of object detection models to bounding box overlap degrees.
Solution

In the model inference phase, the NMS uses the IoU between objects and the corresponding thresholds for processing. In this case, multiple objects with a high overlap degree cannot be detected. The following describes the processing process of the NMS in the traditional object detection model, where B indicates the bounding box.

Input: $B = \{(B_i, S_i)\}, (i=1 \text{ to } N)$, where $S_i$ is the score of $B_i$. $D = \text{Empty set}$

Step 1 Select the box $M$ with the highest score from $B$.
Step 2 Add $M$ and its score to $D$, and delete $M$ and its score from $B$.
Step 3 for each $B_i$ in $B$:
   if $\text{IoU}(M, B_i) \geq NMS\_\text{threshold}$
     Delete $B_i$ and its score from $B$.
   end if
end for
Step 4 Repeat steps 1 to 3 until $B$ is an empty set.
Output: $D$

The disadvantages of the traditional NMS are as follows:

1. The boxes that are greater than the NMS thresholds are all forcibly deleted. The following shows the formula, where $s_i$ indicates the class prediction probability:

   $$s_i = \begin{cases} 
   0, & \text{IoU}(M, B_i) \geq \text{Thresh} \\
   s_i, & \text{IoU}(M, B_i) < \text{Thresh} 
   \end{cases}$$

2. The calculation focuses on the IoU, that is, the overlapping part of two bounding boxes. Other positions of the two bounding boxes are not fully considered.

   The Soft NMS uses a smoother filtering rule to overcome the shortcoming of the traditional NMS.
Another Gaussian expression:

\[
s_i = \begin{cases} 
  s_i(1 - \text{IoU}(M, B_i)), & \text{IoU}(M, B_i) \geq \text{Thresh} \\
  s_i, & \text{IoU}(M, B_i) < \text{Thresh}
\end{cases}
\]

Another Gaussian expression:

\[
s_i = \begin{cases} 
  s_i \frac{\text{IoU}(M, B_i)^2}{\sigma}, & \text{IoU}(M, B_i) \geq \text{Thresh} \\
  s_i, & \text{IoU}(M, B_i) < \text{Thresh}
\end{cases}
\]

The soft NMS can reduce the sensitivity of object detection models to the overlap degrees of bounding boxes.

**Verification**

The open source dataset fruit is used for verification. The following table analyzes the sensitivity to the overlap degrees of bounding boxes before the soft NMS is used.

**Table 6-30** Sensitivity to the overlap degrees of bounding boxes before the soft NMS is used

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>Apple</th>
<th>Banana</th>
<th>Orange</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>1</td>
<td>0.8</td>
<td>0.9167</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>1</td>
<td>0.3333</td>
<td>1</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>1</td>
<td>0.9091</td>
<td>1</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0</td>
<td>0.3811</td>
<td>0.0333</td>
</tr>
</tbody>
</table>

The feature distribution indicates that the entire dataset overlap degree is divided based on percentage intervals, and the corresponding values are recall rates. The sensitivity of Banana is 0.3811.

The following figure shows the change of the sensitivity to bounding box overlap degrees after the soft NMS is used. The sensitivity of the Banana type to bounding box overlap degrees decreases from 0.3811 to 0.2487. This verifies that the algorithm can optimize the model in terms of bounding box overlap.
Table 6-31  Sensitivity to the overlap degrees of bounding boxes after the soft NMS is used

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>Apple</th>
<th>Banana</th>
<th>Orange</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>1</td>
<td>0.8</td>
<td>0.9167</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>1</td>
<td>0.3333</td>
<td>1</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>1</td>
<td>0.9091</td>
<td>1</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0</td>
<td>0.2487</td>
<td>0.0333</td>
</tr>
</tbody>
</table>

Suggestions

In the model inference result, if the detected classes are very sensitive to the overlap degrees of bounding boxes, you are advised to use the soft NMS for model optimization and enhancement during training and inference.

6.4.8.10 Fine-grained Classification Optimization Using Center Loss

Symptom

Fine-grained classification refers to classification tasks where the classes are very similar, such as the types of birds and vehicle styles. In these images, each class has only slight differences. The following figure shows the data of 21 types of sparrows. Each column represents a type, and each type has ten images. These sparrows are highly similar. The features extracted from these similar images using a convolutional neural network (CNN) are usually very similar. In this case, it is difficult to distinguish these similar features using the Softmax Cross-Entropy Loss function.

Figure 6-49 Fine-grained classification
Solution

Center loss

To increase the differentiation between these features, Center loss is proposed in *A Discriminative Feature Learning Approach for Deep Face Recognition*. The principle of this loss function is to set several center points, so that features of different classes are as close as possible to their respective center points. That is, it is expected that the distance within a class decreases and the distance between classes increases. The formula is as follows, where $x$ indicates a feature, $c$ indicates a center, and $c$ is updated with model training.

$$L_C = \frac{1}{2} \sum_{i=1}^{m} \| x_i - c_{y_i} \|^2$$

There are various ways to implement center loss. For details about Pytorch implementation of center loss, visit [https://github.com/KaiyangZhou/pytorch-center-loss](https://github.com/KaiyangZhou/pytorch-center-loss).

class CenterLoss(nn.Module):
    """Center loss."""
    def __init__(self, num_classes=10, feat_dim=2, use_gpu=True):
        super(CenterLoss, self).__init__()
        self.num_classes = num_classes
        self.feat_dim = feat_dim
        self.use_gpu = use_gpu

        if self.use_gpu:
            self.centers = nn.Parameter(torch.randn(self.num_classes, self.feat_dim).cuda())
        else:
            self.centers = nn.Parameter(torch.randn(self.num_classes, self.feat_dim))

    def forward(self, x, labels):
        """Args:
        x: feature matrix with shape (batch_size, feat_dim).
        labels: ground truth labels with shape (batch_size).
        """
        batch_size = x.size(0)
        distmat = torch.pow(x, 2).sum(dim=1, keepdim=True).expand(batch_size, self.num_classes) + \
            torch.pow(self.centers, 2).sum(dim=1, keepdim=True).expand(self.num_classes, batch_size).t() +
            1e-12
        classes = torch.arange(self.num_classes).long()
        if self.use_gpu:
            classes = classes.cuda()
        labels = labels.unsqueeze(1).expand(batch_size, self.num_classes)
        mask = labels.eq(classes.expand(batch_size, self.num_classes))

        dist = distmat * mask.float()
        loss = dist.clamp(min=1e-12, max=1e+12).sum() / batch_size

        return loss

In the preceding project, the MNIST dataset is used for the test. The following figure shows the feature dimension reduction. The first graph in the figure shows
the common Softmax, and the second graph shows the Softmax plus center loss. You can clearly observe that the features are more distinguished.

**Figure 6-50** Feature dimension reduction

Verification

Center loss was initially used in facial recognition tasks. Most open source projects reproduce the scenario on the MNIST dataset. The following describes how center loss works in a natural scenario.

The experiment data comes from the **CUB-200** dataset. 21 types of sparrows are selected for experiment. There are 629 images in the training dataset and 615 images in the test dataset.

The classification network samples ResNet-50 and **main.py** provided by PyTorch are used for the test. The pre-trained ImageNet model is used, the learning rate is set to 0.001, batch-size is set to 32, and 30 epochs are trained.

The classification precision of 73.6% is obtained when only the conventional loss function is used. The following figure shows the visual feature dimension reduction of the last convolutional layer.
The features and labels of the last convolutional layer are input to center loss to calculate the loss value. The loss value and the normal Softmax loss function are weighted (weight: 0.001).

\[
\text{loss} = \text{softmax}\_\text{loss} (\text{pred}\_\text{output}, \text{target}\_\text{label}) + 0.001 \times \text{center}\_\text{loss} (\text{feature}, \text{target}\_\text{label})
\]

In this case, the classification precision is 75.7%. The following figure shows the visual feature dimension reduction of the last layer. According to the classes circled in the figure, the intra-class distance decreases and the inter-class distance increases. However, the effect is not as obvious as that in the test using the MNIST dataset because the background is too complex.

**Figure 6-52 Visual feature dimension reduction of the last layer**

---

**Suggestions**

Center loss can optimize fine-grained classification in natural scenarios. However, the optimization effect will decrease as the background complexity increases.
Convolutional feature visualization can help effectively analyze model performance. This function will be rolled out in the model evaluation module of ModelArts.

6.4.8.11 Common Methods of Optimizing Model Precision in Model Optimization

Overview

In deep learning competitions, many tricks are emerging. One of the controversial methods is to use augmentation during tests to generate multiple copies of the input source images, send them to the models, and combine all inference results. This method is called test time augmentation (TTA). This section describes the TTA principles and suggestions.

Principles

- **TTA process**
  The basic TTA process is as follows: Augment the original images to obtain multiple augmented samples and form a data group with the original images. Use these samples to obtain inference results. Combine the inference results using a certain method to obtain the final inference result and then calculate the precision.

  ![Figure 6-53 TTA process](image)

  The following problems need to be confirmed:
  a. What augmentation method is used to generate samples for the original images?
  b. What method is used to integrate the inference results obtained using the samples?

  The following describes the functions of the TTA and how to use the TTA by using the functions provided by the ModelArts platform.

- **Example of using the TTA**
  - Dataset: The following figure shows a dataset example. The left part consists of 754 normal images. The right part consists of 358 abnormal
electrical board images. After some augmentation measures are taken, the number of normal images increases to 1508, and the number of abnormal images decreases to 1432.

**Figure 6-54** Dataset example

- **Framework and algorithm**: See [ImageNet open source code](#).
- **Training policy**: 50 epochs, initial learning rate $lr_{0.001}$, batchsize16 trained using Adam’s optimizer

Model precision

<table>
<thead>
<tr>
<th>Precision</th>
<th>Normal</th>
<th>Abnormal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>97.2%</td>
<td>71.3%</td>
</tr>
<tr>
<td>Accuracy</td>
<td>89.13%</td>
<td></td>
</tr>
</tbody>
</table>

**TTA process**

a. Select an augmentation method to obtain the samples. You can select the method as follows:
   i. Select the augmentation method used in training.

   For example, in the ImageNet training code provided by PyTorch, the operator `transforms.RandomHorizontalFlip()` is used for vertical flipping. For the model, there are many images that are vertically flipped. Therefore, you can use vertical flipping as an augmentation method.

   ii. Evaluate the model and analyze the augmentation method to be used based on the model evaluation result.

   Evaluate the original model. The evaluation code is as follows (the evaluation code is obtained by modifying the code for performing forward inference in the validation section of the open source code):

   ```python
   with torch.no_grad():
       end = time.time()
   for i, (images, target) in enumerate(val_loader):
       if args.gpu is not None:
           images = images.cuda(args.gpu, non_blocking=True)
           target = target.cuda(args.gpu, non_blocking=True)
       # compute output
   ```
The evaluation requires three lists. The logits results are combined into the pred_list, which stores the prediction result of each image, for example, `[8.725419998168945, 21.92235565185547]...[xxx, xxx]`. The target_list consists of the labels of each image, for example, `[0, 1, 0, 1, 1..., 1, 0]`. The name_list consists of the paths for storing original image files, for example, `[xxx.jpg, ... xxx.jpg]`. The analyse interface in the deep_moxing library is called to generate a `model_analysis_results.json` file in `save_path`. The file is uploaded to the output directory of any training task on the page. Then the model evaluation result is displayed on the evaluation page.

**Figure 6-55** Viewing the evaluation result

The model sensitivity needs to be analyzed.
Table 6-32 Analysis on the sensitivity of a model to image clarity

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>0.7929</td>
<td>0.8727</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>0.8816</td>
<td>0.7429</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>0.9363</td>
<td>0.7229</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>0.9462</td>
<td>0.7912</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>0.9751</td>
<td>0.7619</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.0643</td>
<td>0.0523</td>
</tr>
</tbody>
</table>

As shown in the preceding figure, the F1 score of class 0 (normal class) increases with the image clarity. That is, the model performs better in detecting the normal class on clear images. The precision for detecting class 1 (abnormal class) decreases as the image clarity increases. Blurred images can make the model more accurate to detect abnormal classes. Because this model focuses on the identification of abnormal classes, image blurring can be used as a TTA method.

b. Add the TTA to PyTorch.

The advantage of PyTorch is that you can directly obtain the tensor before model input and perform the required operation, for example, verification.

```python
with torch.no_grad():
    end = time.time()
    for i, (images, target) in enumerate(val_loader):
        if args.gpu is not None:
            images = images.cuda(args.gpu, non_blocking=True)
```

The images obtained here are the image data of a batch that has been pre-processed. Two augmentation methods are determined in `a`: vertical flipping and blurring.

If the version is later than 0.4.0, you can use the following code for flipping in PyTorch:

```python
def flip(x, dim):
    indices = [slice(None)] * x.dim()
    indices[dim] = torch.arange(x.size(dim) - 1, -1, -1, dtype=torch.long, device=x.device)
    return x[tuple(indices)]
```

`dim` indicates the mode. In this example, 2 indicates vertical flipping, 3 indicates horizontal flipping, and 1 indicates channel rotation. Use `img_flip = flip(images, 2)` to obtain the images that are flipped vertically.

You can use the blur operations provided by CV2 to perform blur operations.

```python
img = images.numpy()
img(0) = cv2.blur(img[0], (3, 3))
images_blur = torch.from_numpy(img.copy())
```
c. Combine the results.

Three outputs are obtained: origin_result (inference results of the original images), flip_output (results obtained after vertical flipping), and blur_output (results obtained after blurring).

How are they combined?

For flip_output, what is the proportion of flipped images in the original training? What is the contribution weight of a flipped image to the result in the final output? As many students with deep learning experience know, the flipping probability is 0.5. That is, the proportion of flipped images is about 0.5. The final contribution of the flipped image is 0.5. The following formula can be obtained:

\[
\text{logits} = 0.5 \times \text{origin_result} + 0.5 \times \text{flip_result}
\]

In this case, the precision of the model is as follows:

<table>
<thead>
<tr>
<th>Operation</th>
<th>ACC</th>
<th>Recall of Normal Class</th>
<th>Recall of Abnormal Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Originals</td>
<td>89.13%</td>
<td>97.2%</td>
<td>71.3%</td>
</tr>
<tr>
<td>Flipping result combining</td>
<td>87.74%</td>
<td>93.7%</td>
<td>72.7%</td>
</tr>
</tbody>
</table>

Although the accuracy of the normal class decreases, the recall of the abnormal class increases.

For blur_output, the precision of the abnormal class is the highest when the value is between 0 and 20%. However, the precision of the normal class drops. In addition, blurring is used to improve the precision of the abnormal class. Therefore, assume that the contribution of the blurred image is 0.5, the formula is as follows:

\[
\text{logits} = 0.5 \times \text{origin_result} + 0.5 \times \text{blur_output}
\]

In this case, the precision of the model is as follows:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Accuracy</th>
<th>Recall of Normal Class</th>
<th>Recall of Abnormal Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Originals</td>
<td>89.13%</td>
<td>97.2%</td>
<td>71.3%</td>
</tr>
<tr>
<td>Blurring result combining</td>
<td>88.117%</td>
<td>94.8%</td>
<td>73.3%</td>
</tr>
</tbody>
</table>
In the preceding table, the accuracy of the normal class decreases greatly, and that of the abnormal class increases significantly, which is consistent with the analysis result of model evaluation.

In conclusion, the adjustment causes a large loss to the normal class and the overall precision decreases. However, this is consistent with the model analysis result. The adjustment aims to improve the recall of the abnormal class. The model evaluation result is slightly better than the result by augmenting the original images.

**Summary**

In this test, two TTA methods are used. One is to use the built-in augmentation method. The other one is to analyze model sensitivity and determine the image feature interval that is the most helpful for model inference. Of course, the TTA will increase the model inference time. For AI algorithms that have demanding requirements on the inference latency, you need to carefully select a proper solution.

**6.4.8.12 Data Augmentation Based on Model Evaluation**

**Scenario**

Datasets are critical for model training. The ModelArts platform requires that each class to be trained must contain at least five images. However, datasets usually need to be expanded in industrial-grade scenarios. This section describes a method for expanding a dataset with a label.

**Principles**

**Dataset details**

This is a classification problem. The model must identify the defects on the surface of industrial parts and determine whether they are defective parts. The following figure shows an example.

**Figure 6-56** Defects on the surface of industrial parts

This figure shows the surfaces of two PV modules. The left one is normal, and the right one is abnormal. A model is required to distinguish the two classes of images
to help locate the abnormal PV modules. There are 754 normal samples and 358 abnormal samples. The validation set has 754 normal samples and 357 abnormal samples. There are about 2000 samples in total, which are far less than the samples required to achieve accuracy of 95% or higher. Use PyTorch to load the ResNet50 model for training based on this dataset. The overall precision is about 86.06%. The recall of normal classes is 97.3%, and that of abnormal classes is 62.9%. The result does not meet the customer's expectation.

Few-Shot Learning (FSL) is a common method of feeding a learning model with a very small amount of training data. It focuses on data and model training, that is, features extracted from images. In this dataset, the images are 300 x 300 grayscale images and are the front view images of the PV module surfaces. They are efficiently pre-processed images. Flipping this type of images has little impact on the overall image structure. Therefore, flip these images to increase data diversity. The following figure shows the flipping result.

**Figure 6-57 Flipping result**

In this way, the number of images in the dataset increases from 1100 to 2200. There is no other way available for expanding the dataset. In this case, you need to use the ModelArts model evaluation function to evaluate the data generalization capability of the model. Call the model evaluation API, that is, the `analyse` API under `deep_moxing.model_analysis`.

```python
def validate(val_loader, model, criterion, args):
batch_time = AverageMeter('Time', ':6.3f')
losses = AverageMeter('Loss', ':.4e')
top1 = AverageMeter('Acc@1', ':6.2f')
top5 = AverageMeter('Acc@5', ':6.2f')
progress = ProgressMeter(
    len(val_loader),
    [batch_time, losses, top1, top5],
    prefix='Test: ')
pred_list = []
target_list = []
# switch to evaluate mode
model.eval()
with torch.no_grad():
    end = time.time()
    for i, (images, target) in enumerate(val_loader):
        if args.gpu is not None:
            images = images.cuda(args.gpu, non_blocking=True)
        target = target.cuda(args.gpu, non_blocking=True)
        # compute output
        output = model(images)
        loss = criterion(output, target)
        # measure accuracy and record loss
        prec1, prec5 = accuracy(output, target, topk=(1, 5))
        losses.update(loss.item(), images.size(0))
```

---

**Image:** Flipping result

In this way, the number of images in the dataset increases from 1100 to 2200. There is no other way available for expanding the dataset. In this case, you need to use the ModelArts model evaluation function to evaluate the data generalization capability of the model. Call the model evaluation API, that is, the `analyse` API under `deep_moxing.model_analysis`.
```python
# compute output
output = model(images)
loss = criterion(output, target)
# Obtain the model inference result pred and the actual target target.
pred_list += output.cpu().numpy()[:, :2].tolist()
target_list += target.cpu().numpy().tolist()
# measure accuracy and record loss
acc1, acc5 = accuracy(output, target, topk=(1, 5), i=i)
losses.update(loss.item(), images.size(0))
top1.update(acc1[0], images.size(0))
top5.update(acc5[0], images.size(0))
# measure elapsed time
batch_time.update(time.time() - end)
end = time.time()
if i % args.print_freq == 0:
    progress.display(i)
    # TODO: this should also be done with the ProgressMeter
    print(' * Acc@1 {top1.avg:.3f} Acc@5 {top5.avg:.3f}'
          .format(top1=top1, top5=top5))
# Obtain the image storage path name.
namelist = val_loader.dataset.samples
for idx in range(len(namelist)):
    namelist[idx] = namelist[idx][0]
analyse(task_type='image_classification', save_path='/home/image_labeled/',
         pred_list=pred_list, label_list=target_list, name_list=namelist)
return top1.avg
```

Most of the preceding code is the validation code in the PyTorch training ImageNet. Obtain three lists: `pred` (model reference results), `target` (actual image classes), and `name` (image storage paths). Call the analyse API according to the preceding code. A JSON file is generated in the `save_path` directory and saved to the ModelArts training output directory. Then, you can view the model analysis result in the evaluation result. You can view the result visually after uploading the JSON file generated offline to the online directory. Table 6-35 and Table 6-36 list the sensitivity analysis results.

**Table 6-35** Analysis on the sensitivity of a model to image brightness

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>0.7273</td>
<td>0.8864</td>
</tr>
<tr>
<td>20% - 40%</td>
<td>0.8446</td>
<td>0.6892</td>
</tr>
<tr>
<td>40% - 60%</td>
<td>0.9077</td>
<td>0.4615</td>
</tr>
<tr>
<td>60% - 80%</td>
<td>0.9496</td>
<td>0.5116</td>
</tr>
<tr>
<td>80% - 100%</td>
<td>0.966</td>
<td>0.5625</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.0864</td>
<td>0.1516</td>
</tr>
</tbody>
</table>

**Table 6-36** Analysis on the sensitivity of a model to image brightness

<table>
<thead>
<tr>
<th>Feature Distribution</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 20%</td>
<td>0.7556</td>
<td>0.8333</td>
</tr>
</tbody>
</table>
The preceding two tables indicate the test precision of images with different feature value ranges. For example, the test precision of class 0 when the brightness range is 0%-20% is much lower than that when the brightness is in other ranges. This model is used to detect images of class 1. Images of class 1 are sensitive to both brightness and clarity. That is, the model cannot properly process the images whose brightness and clarity change.

ModelArts provides the data augmentation function to directly expand datasets.

After data augmentation, the dataset contains 2,210 normal images and 1,174 abnormal images. Import the dataset into PyTorch for training. The result is as follows:

### Table 6-37 Training results after data augmentation

<table>
<thead>
<tr>
<th>Item</th>
<th>accuracy</th>
<th>Recall of Normal Class</th>
<th>Recall of Abnormal Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Originals</td>
<td>86.06%</td>
<td>97.3%</td>
<td>62.9%</td>
</tr>
<tr>
<td>Images expanded from 1,100 to 2,940</td>
<td>86.31%</td>
<td>97.6%</td>
<td>62.5%</td>
</tr>
</tbody>
</table>

The preceding results show that the precision does not improve significantly. Analyze the dataset again. The datasets for industry scenarios often have uneven samples. Although the ratio is close to 2:1 in this dataset, the detection requirements are high for the abnormal class. Therefore, the model should be oriented to the abnormal class. The results of class 1 indicate that the normal and abnormal samples are unbalanced. Therefore, the following two augmentation methods are used only for the abnormal class, that is, the abnormal images. Finally, about 3000 images (1508 normal images and 1432 abnormal images) are obtained. In this way, the samples are balanced. Import the model to ResNet50 for training. The following table lists the precision.
### Table 6-38 Results after data modification

<table>
<thead>
<tr>
<th>Item</th>
<th>accuracy</th>
<th>Recall of Normal Class</th>
<th>Recall of Abnormal Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Originals</td>
<td>86.06%</td>
<td>97.3%</td>
<td>62.9%</td>
</tr>
<tr>
<td>Images expanded from 1,100 to 2,940</td>
<td>89.13%</td>
<td>97.2%</td>
<td>71.3%</td>
</tr>
</tbody>
</table>

#### Summary

In the validation set, Accuracy 1 increases by nearly 3% and the recall increases by 8.4% when there are 754 normal samples and 357 abnormal samples. Therefore, direct dataset expansion is effective. Model evaluation is helpful for you to choose an effective augmentation method. It is important to eliminate the problems of the original dataset, for example, unbalanced samples.

Based on the results of this test, we modify some training policies and use a more powerful network. Our model training results meet the customer's requirements.

Dataset reference:


#### 6.4.8.13 Model Attack Based on Sample Preprocessing

**Scenario**

Model robustness has always been a research focus, and there have been many excellent practices of model attack defense. However, most model attacks require the models, which is impossible in most cases. The model attack method described in this section focuses on preprocessing (the resizing phase), which is necessary for all models. This method comes from Seeing is Not Believing: Camouflage Attacks on Image Scaling Algorithms.

**Principles**

Most models have requirements on the size of input samples, for example, 224 x 224 or a custom size. Moreover, these sizes are enumerable and exhaustive, which make black-box attacks possible. Before a sample enters a model, the sample is
preprocessed. The basic operation is to resize the sample to the size required by the model,

which causes information loss. If most reliable information is lost but attack information is retained during model resizing, an attack may be launched.

There are several common sample resizing algorithms, including nearest neighbor interpolation (INTER_NEAREST), bilinear interpolation (INTER_LINEAR), bicubic interpolation (INTER_CUBIC), Lanczos interpolation (INTER_LANCZOS4), and local pixel-based resampling (INTER_AREA). The nearest neighbor interpolation algorithm is also a sampling algorithm. If it is used to resize a sample, the pixels that are reserved during sampling may be calculated based on the size of original samples and the size of attack images. You only need to place the attack images on these pixels to hide the attack information on the images that look normal. The logic of bilinear interpolation is similar.

**Figure 6-58** Attack effect

![Attack effect](image1)

**Figure 6-59** Effect after resizing

![Effect after resizing](image2)
The larger the original samples, the better the attack effect. Partial modification is allowed. For example, a certificate sample can be attacked only by modifying the key words.

**Defense**

This attack method has obvious advantages and disadvantages.

- **Advantage:** This method is simple, and can be used for black-box attacks. A smaller model input sample size means the model is more vulnerable to attacks. A larger difference between the attack sample size and the model input sample size means a better attack effect.
- **Disadvantage:** This attack is easy to defend.
- **Defense method:** During resizing, use the more complex Lanczos interpolation algorithm (INTER_LANCZOS4), which may compromise some performance but can directly prevent such attacks. Alternatively, randomly tailor the samples and then resize them. In this case, the attack pixels that have been specially arranged will not be selected.
7 Model Deployment

7.1 Model Deployment Overview

After a training job is complete and a model is generated, you can deploy the model on the Service Deployment page. You can also deploy the model imported from OBS. ModelArts supports the following deployment types:

- **Real-Time Services**
  Deploy a model as a web service to provide real-time test UI and monitoring capabilities.

- **Batch Services**
  A batch service can perform inference on batch data. After data processing is complete, the batch service automatically stops.

7.2 Real-Time Services

7.2.1 Deploying a Model as a Real-Time Service

After a model is prepared, you can deploy the model as a real-time service and predict and call the service.

**NOTE**

A maximum of 20 real-time services can be deployed by a user.

**Prerequisites**

- Data has been prepared. Specifically, you have created a model in the Normal state in ModelArts.
- Ensure that the account is not in arrears. Resources are consumed when services are running.
Procedure

1. Log in to the ModelArts management console. In the left navigation pane, choose **Service Deployment > Real-Time Services**. By default, the system switches to the **Real-Time Services** page.

2. In the real-time service list, click **Deploy** in the upper left corner. The **Deploy** page is displayed.

3. Set parameters for a real-time service.
   a. Enter basic information about model deployment. For details about the parameters, see **Table 7-1**.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Billing Mode</td>
<td>Currently, only pay-per-use billing is supported.</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the real-time service. Set this parameter as prompted.</td>
</tr>
<tr>
<td>Auto Stop</td>
<td>After this parameter is enabled and the auto stop time is set, a service automatically stops at the specified time. If this parameter is disabled, a real-time service keeps running and billing. The function can help you avoid unnecessary billing. The auto stop function is enabled by default, and the default value is <strong>1 hour later</strong>. Currently, the options are <strong>1 hour later</strong>, <strong>2 hours later</strong>, <strong>4 hours later</strong>, <strong>6 hours later</strong>, and <strong>Custom</strong>. If you select <strong>Custom</strong>, you can enter any integer from 1 to 24 hours in the textbox on the right.</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description of the real-time service.</td>
</tr>
</tbody>
</table>

   b. Enter key information including the resource pool and model configurations. For details, see **Table 7-2**.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sub-Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Pool</td>
<td>Public resource pools</td>
<td>Instances in the public resource pool can be of the CPU or GPU type. Pricing standards for resource pools with different instance flavors are different. For details, see <strong>Product Pricing Details</strong>. Currently, the public resource pool only supports the pay-per-use billing mode.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Sub-Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Model and Configuration</td>
<td>Model Source</td>
<td>You can select <strong>My Models</strong> or <strong>My Subscriptions</strong> based on site requirements. The models that match the model sources are displayed.</td>
</tr>
<tr>
<td></td>
<td>Model</td>
<td>The system automatically associates with the list of available models. Select a model in the <strong>Normal</strong> status and its version.</td>
</tr>
<tr>
<td></td>
<td>Traffic Ratio (%)</td>
<td>Set the traffic proportion of the current instance node. Service calling requests are allocated to the current version based on this proportion. If you deploy only one version of a model, set this parameter to <strong>100%</strong>. If you select multiple versions for gated launch, ensure that the sum of the traffic ratios of multiple versions is <strong>100%</strong>.</td>
</tr>
<tr>
<td></td>
<td>Specifications</td>
<td>If you select <strong>Public resource pools</strong>, you can select the CPU or GPU resources based on site requirements. For details, see <strong>Table 7-3</strong>.</td>
</tr>
<tr>
<td></td>
<td>Compute Nodes</td>
<td>Set the number of instances for the current model version. If you set <strong>Instances</strong> to <strong>1</strong>, the standalone computing mode is used. If you set <strong>Instances</strong> to a value greater than <strong>1</strong>, the distributed computing mode is used. Select a computing mode based on the actual requirements.</td>
</tr>
<tr>
<td></td>
<td>Environment Variable</td>
<td>Set environment variables and inject them to the container instance.</td>
</tr>
<tr>
<td></td>
<td>Add Model and Configuration</td>
<td>ModelArts supports multiple model versions and flexible traffic policies. You can use gated launch to smoothly upgrade the model version. <strong>NOTE</strong> If the selected model has only one version, the system does not display <strong>Add Model Version and Configuration</strong>.</td>
</tr>
<tr>
<td></td>
<td>Data Collection</td>
<td>Disabled by default. To enable this function, see <strong>Collecting Data</strong> for details and set the parameters as required.</td>
</tr>
<tr>
<td></td>
<td>Hard Example Filtering</td>
<td>Disabled by default. To enable this function, see <strong>Collecting Data</strong> for details and set the parameters as required.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Sub-Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Application Authentication</td>
<td>Application</td>
<td>Disabled by default. To enable this function, see <strong>Accessing a Real-Time Service (Application Authentication)</strong> for details and set the parameters as required.</td>
</tr>
</tbody>
</table>

**Table 7-3 Supported specifications**

<table>
<thead>
<tr>
<th>Specifications</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExeML (CPU)</td>
<td>Only be used by models trained in ExeML projects.</td>
</tr>
<tr>
<td>ExeML (GPU)</td>
<td>Only be used by models trained in ExeML projects.</td>
</tr>
<tr>
<td>CPU: 2 vCPUs</td>
<td>8 GiB</td>
</tr>
<tr>
<td>CPU: 2 vCPUs</td>
<td>8 GiB</td>
</tr>
<tr>
<td>CPU: 8 vCPUs</td>
<td>32 GiB</td>
</tr>
</tbody>
</table>

**Figure 7-1 Setting model information**

4. After confirming the entered information, complete service deployment as prompted. Generally, service deployment jobs run for a period of time, which may be several minutes or tens of minutes depending on the amount of your selected data and resources.

**NOTE**

After a real-time service is deployed, it is started immediately. During the running, you will be charged based on your selected resources.

You can go to the real-time service list to view the basic information about the real-time service. In the real-time service list, after the status of the newly deployed service changes from **Deploying** to **Running**, the service is deployed successfully.
7.2.2 Viewing Service Details

After a model is deployed as a real-time service, you can access the real-time service page to view the service details.

1. Log in to the ModelArts management console and choose **Service Deployment > Real-Time Services**.

2. On the **Real-Time Services** page, click the name of the target service. The service details page is displayed.

   You can view the service name and status. You can click the edit icon on the right of **Description** to edit the description.

   **Figure 7-2 Service details page**

3. You can switch between tabs on the details page of a real-time service to view more details. For details, see **Table 7-4**.

**Table 7-4 Service details**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage Guides</td>
<td>Displays the API address, model information, input parameters, and output parameters. You can click to copy the API address to call the service. If application authentication is supported, you can view the API address and authorization management details, including the application name, AppKey, and AppSecret, in the Usage Guides. You can also add or cancel authorization for an application.</td>
<td></td>
</tr>
<tr>
<td>Prediction</td>
<td>Performs a prediction test on the real-time service. For details, see Testing a Service.</td>
<td></td>
</tr>
<tr>
<td>Configuration Updates</td>
<td>Displays <strong>Existing Configuration</strong> and <strong>Historical Updates</strong>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Existing Configuration</strong>: includes the model name, version, status, traffic ratio, compute node flavor, and compute node count.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Historical Updates</strong>: displays historical model information.</td>
<td></td>
</tr>
</tbody>
</table>
### Parameter | Description
--- | ---
Monitoring | Displays **Resource Usage** and **Model Calls**.
- **Resource Usage**: includes the used and available CPU, memory, and GPU resources.
- **Model Calls**: indicates the number of model calls. The statistics collection starts after the model status changes to **Ready**.

Logs | Displays the log information about each model in the service. You can view logs generated in the latest 5 minutes, latest 30 minutes, latest 1 hour, and user-defined time segment.
- You can select the start time and end time when defining the time segment.

Sharing | Displays the sharing information about the service, including the users who have subscribed to the service and the number of service calls.

### 7.2.3 Testing a Service

After a model is deployed as a real-time service, you can debug code or add files for testing on the **Prediction** tab page. Based on the input request (JSON text or file) defined by the model, the service can be tested in either of the following ways:

1. **JSON Text Prediction**: If the input type of the model of the deployed service is JSON text, that is, the input does not contain files, you can enter the JSON code on the **Prediction** tab page for service testing.

2. **File Prediction (Images and Audios)**: If the input type of the model of the deployed service is file, including images, audios, and videos, you can add images on the **Prediction** tab page for service testing.

**NOTE**

If the input type is image, the size of a single image must be less than 10 MB.

**Input Parameters**

For the service that you have deployed, you can learn about its input parameters of the service, that is, the input request type mentioned above, on the **Usage Guides** tab page of the service details page.
The input parameters displayed on the **Usage Guides** tab page depend on the model source that you select.

- If your model comes from ExeML or a built-in algorithm, the input and output parameters are defined by ModelArts. For details, see the **Usage Guides** tab page. On the **Prediction** tab page, enter the corresponding JSON text or file for service testing.

- If you use a custom model, that is, the inference code and the configuration file are compiled by yourself (**Specifications for Compiling the Model Configuration File**), the **Usage Guides** tab page only visualizes your configuration files. The following figure shows the mapping between the input parameters displayed on the **Usage Guides** tab page and the configuration file.

**Figure 7-4** Mapping between the configuration file and **Usage Guides**

- If your model is imported using a model template, the input and output parameters vary with the template. For details, see the description in **Model Template Overview**.

**JSON Text Prediction**

1. Log in to the ModelArts management console and choose **Service Deployment** > **Real-Time Services**.

2. On the **Real-Time Services** page, click the name of the target service. The service details page is displayed. On the **Prediction** tab page, enter the prediction code and click **Predict** to perform prediction.
File Prediction (Images and Audios)

1. Log in to the ModelArts management console and choose Service Deployment > Real-Time Services.

2. On the Real-Time Services page, click the name of the target service. The service details page is displayed. On the Prediction tab page, click Upload and select a test file. After the file is uploaded successfully, click Predict to perform a prediction test. In Figure 7-5, the label, position coordinates, and confidence score are displayed.

![Figure 7-5 Image prediction](image)

7.2.4 Accessing a Real-Time Service (Token-based Authentication)

If a real-time service is in the Running state, the real-time service has been deployed successfully. This service provides a standard RESTful API for users to call. Before integrating the API to the production environment, commission the API. You can use either of the following methods to send an inference request to the real-time service:

**Method 1: Use GUI-based Software for Inference (Postman)**

**Method 2: Run the cURL Command to Send an Inference Request**

**Method 1: Use GUI-based Software for Inference (Postman)**

1. Download Postman and install it, or install the Postman Chrome extension. Alternatively, use other software that can send POST requests. Postman 7.24.0 is recommended.

2. Open Postman. Figure 7-6 shows the Postman interface.
3. Set parameters on Postman. The following uses image classification as an example.

- Select a POST task and copy the API URL to the POST text box. To obtain the API URL of the real-time service, switch to the **Usage Guides** tab on the page providing details about the real-time service. On the **Headers** tab page, set **Key** to **X-Auth-Token** and **Value** to the obtained token.

  For details about how to obtain a token, see **Obtaining a User Token**.

  The real-time service APIs generated by ModelArts do not support tokens whose scope is domain. Therefore, you need to obtain the token whose scope is project.

  **NOTE**
  
  You can also use the AK and SK to encrypt API calling requests. For details, see **SDK Reference > Session Authentication > AK/SK-based Authentication**.

**Figure 7-7 Parameter settings**

- On the **Body** tab page, file input and text input are available.

  **File input**

  Select **form-data**. Set **KEY** to the input parameter of the model, for example, **images**. Set **VALUE** to an image to be inferred (only one image can be inferred). See **Figure 7-8**.

**Figure 7-8 Setting parameters on the **Body** tab page**
- **Text input**

Select raw and then JSON(application/json). Enter the request body in the text box below. An example request body is as follows:

```json
{
    "meta": {
        "uuid": "10eb0091-887f-4839-9929-cbc884f1e20e"
    },
    "data": {
        "req_data": [
            {
                "sepal_length": 3,
                "sepal_width": 1,
                "petal_length": 2.2,
                "petal_width": 4
            }
        ]
    }
}
```

**meta** can carry a universally unique identifier (UUID). When you call an API, the system provides a UUID. When the inference result is returned, the UUID is returned to trace the request. If you do not need this function, leave **meta** blank. **data** contains a **req_data** array for one or multiple pieces of input data. The parameters of each piece of data are determined by the model, for example, **sepal_length** and **sepal_width** in this example.

4. After setting the parameters, click **Send** to send the request. The result is displayed in the response.

- Inference result using file input: Figure 7-9 shows an example. The field values in the return result vary with the model.

- Inference result using text input: Figure 7-10 shows an example. The request body contains **meta** and **data**. If the request contains **uuid**, **uuid** will be returned in the response. Otherwise, **uuid** is left blank. **data** contains a **resp_data** array for the inference results of one or multiple pieces of input data. The parameters of each result are determined by the model, for example, **sepal_length** and **predictresult** in this example.
Figure 7-9 File inference result

```
POST https://modelarts-infer-1.cn-north-1.myhuaweicloud.com/v1/infers/cb3c634-3e65-4750-e00c-55c6553e0381

<table>
<thead>
<tr>
<th>Param</th>
<th>Authorization</th>
<th>Headers</th>
<th>Body</th>
<th>Pre-request Script</th>
<th>Texts</th>
</tr>
</thead>
<tbody>
<tr>
<td>form-data</td>
<td>x-www-form-urlencoded</td>
<td>raw</td>
<td>binary</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**KEY**

<table>
<thead>
<tr>
<th>images</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose File</td>
<td>image_0001.jpg</td>
</tr>
</tbody>
</table>

**Body**

```
```json
{
  "confidences": [
    0.37127092488489465,
    0.2555185956903645,
    0.14066125971389807,
    0.2811202260570255675,
    0.83250678964907673
  ],
  "logits": [
    1.1059842499869965,
    0.782396660865954995,
    -1.2095318865934966,
    -0.603358975060956,
    -1.4578913594717627,
    0.737247884273829
  ],
  "labels": [
    0,
    1,
    5,
    3,
    2
  ]
}
```
Method 2: Run the cURL Command to Send an Inference Request

The command for sending inference requests can be input as a file or text.

1. **File input**

```
curl -F 'images=@Image path' -H 'X-Auth-Token: Token value' -X POST Real-time service URL
```

- `-F` indicates file input. In this example, the parameter name is `images`, which can be changed as required. The image storage path follows @.
- `-H` indicates the header of the POST command. `X-Auth-Token` is the KEY value on the Headers page. `Token value` indicates the obtained token. For details about how to obtain the token, see Authentication.
- `POST` is followed by the API URL of the real-time service.

The following is an example of the cURL command for inference with file input:
```
curl -F 'images=@/home/data/test.png' -H 'X-Auth-Token:MIISkAY***80T9wHQ==' -X POST https://modelarts-infers-1.cn-north-1.myhuaweicloud.com/v1/infers/eb3e0c54-3dfa-4750-af0c-95c45e5d3e83
```

2. **Text input**

```
curl -d '{"data":{"req_data":["sepal_length":3,"sepal_width":1,"petal_length":2.2,"petal_width":4]}}' -H 'X-Auth-Token:MIISkAY***80T9wHQ== -H 'Content-type: application/json' -X POST https://modelarts-infers-1.cn-north-1.myhuaweicloud.com/v1/infers/eb3e0c54-3dfa-4750-af0c-95c45e5d3e83
```

- `-d` indicates the text input of the request body.
7.2.5 Accessing a Real-Time Service (AK/SK-based Authentication)

If a real-time service is in the Running state, the real-time service has been deployed successfully. This service provides a standard RESTful API for users to call. Users can call the API using AK/SK-based authentication.

When AK/SK-based authentication is used, you can use the APIG SDK or ModelArts SDK to access real-time services. For details about how to use the ModelArts SDK to access real-time services, see AK/SK-based Authentication. This section describes how to use the APIG SDK to access a real-time service. The process is as follows:

1. Obtaining an AK/SK Pair
2. Obtaining the URI
3. Preparing Environment (Python)
4. API Calling Example

**NOTE**

1. AK/SK-based authentication supports API requests with a body not larger than 12 MB. For API requests with a larger body, token-based authentication is recommended.
2. The client must synchronize its local time with the NTP server to avoid a large offset in the value of X-Sdk-Date in the request header. API Gateway checks the time format and compares the time with the time when API Gateway receives the request. If the time difference exceeds 15 minutes, API Gateway will reject the request.

**Obtaining an AK/SK Pair**

If an AK/SK pair is already available, skip this step. Find the downloaded AK/SK file, which is usually named credentials.csv.

As shown in the following figure, the file contains the username, AK, and SK.

**Figure 7-11 Content of the credential.csv file**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>UserName</td>
<td>Access Key Id</td>
</tr>
<tr>
<td>2</td>
<td>hu****dg</td>
<td>QITWA****UT2QVKYUC</td>
</tr>
</tbody>
</table>

Perform the following operations to generate an AK/SK pair:

1. Register with and log in to the management console.
2. Click the username and choose My Credentials from the drop-down list.
4. Click Create Access Key. The Identity Verification dialog box is displayed.
5. Enter the SMS verification code and click OK to download the access key. Keep the key secure.
Obtaining the URI

Before calling an API, you need to obtain the API URI. The procedure is as follows:

1. Log in to the ModelArts management console. In the left navigation pane, choose Service Deployment > Real-Time Services. By default, the system switches to the Real-Time Services page.
2. Click the name of the target service. The service details page is displayed.
3. Obtain the API of the service.

Preparing Environment (Python)

This section uses Python as an example to describe how to use the APIG SDK. For details about how to download and use the SDK in other languages, see the API Gateway Developer Guide.

1. Obtain the Python installation package (version 2.7.9 or 3.x) from the Python official website and install it. After Python is installed, you can run the pip install requests command to install the requests library using the Python package manager pip.
If a certificate error occurs when you use pip to install the requests library, download and execute the file using Python to upgrade pip. Then run the preceding command to install the library.

2. Obtain IntelliJ IDEA from the IntelliJ IDEA official website and install it. Install the Python plug-in on IntelliJ IDEA. See Figure 7-14.

![Figure 7-14 Installing the Python plug-in](image)

3. Download the ApiGateway-python-sdk.zip. The following table describes the directory structure of the decompressed package.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apig_sdk_<em>init</em>_.py</td>
<td>SDK code</td>
</tr>
<tr>
<td>apig_sdk\signer.py</td>
<td>Sample code</td>
</tr>
<tr>
<td>main.py</td>
<td>Sample code for the backend signature</td>
</tr>
<tr>
<td>backend_signature.py</td>
<td></td>
</tr>
<tr>
<td>licenses\license-requests</td>
<td>Third-party library license file</td>
</tr>
</tbody>
</table>

4. Create a project.
b. Click **Next**. The following page is displayed. Click **...**, select the path for the decompressed SDK package, and click **Finish**.

**Figure 7-15** Creating a project

![Creating a project](image)

5. After the project is created, the directory structure is shown in the following figure. **main.py** is sample code. Modify the parameters to suit your requirements before use. For details about the sample code, see **API Calling Example**.
**Figure 7-17** Directory structure of a new project

```python
directory
  \./.idea
  \./ApiGateway-python-sdk
    \./api_gsdk
      \./_init_.py
      \./signer.py
    \./licenses
    \./main.py
    \./test_signer.py
    \./python.ml
```

**API Calling Example**

1. Add `apig_sdk` to the project.
   ```python
   from apig_sdk import signer
   import requests
   ```

2. Generate a new signer and enter the AK and SK. For details about how to obtain the AK and SK, see *Obtaining an AK/SK Pair*.
   ```python
   sig = signer.Signer()
   sig.Key = "UATBQ1PQ0150NVCDAA"
   sig.Secret = "ap6H7L******0QvHCNk"
   ```

3. Generate a request object and specify the method, request URI, header, and body.
   ```python
   r = signer.HttpRequest(method, uri, header, body)
   ```

**Table 7-6 HttpRequest parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sub-Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>N/A</td>
<td>Yes</td>
<td>The value can be <strong>GET, POST, PUT, or DELETE.</strong></td>
</tr>
<tr>
<td>uri</td>
<td>N/A</td>
<td>Yes</td>
<td>Enter the API URI of the real-time service. For details about how to obtain the URI, see <em>Obtaining the URI</em>.</td>
</tr>
<tr>
<td>header</td>
<td>x-stage</td>
<td>Yes</td>
<td>API release environment, which can only be <strong>RELEASE</strong>.</td>
</tr>
<tr>
<td></td>
<td>Content-Type</td>
<td>No</td>
<td>Content type, which can only be <strong>application/json</strong>. For details about the <strong>multipart/form-data</strong> request body, see <em>Table 7-7</em>.</td>
</tr>
</tbody>
</table>
### Table 7-7 files parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request parameter</td>
<td>Enter the parameter name of the real-time service.</td>
</tr>
<tr>
<td>Load path</td>
<td>Path in which the file is stored.</td>
</tr>
<tr>
<td>File content</td>
<td>Content of the file to be uploaded.</td>
</tr>
<tr>
<td>File type</td>
<td>Type of the file to be uploaded, which can be one of the following options:</td>
</tr>
<tr>
<td></td>
<td>● <code>txt</code>: text/plain</td>
</tr>
<tr>
<td></td>
<td>● <code>jpg/jpeg</code>: image/jpeg</td>
</tr>
<tr>
<td></td>
<td>● <code>png</code>: image/png</td>
</tr>
<tr>
<td>Request header</td>
<td>You are advised to set this parameter to <code>{}</code>. The request header is specified by <code>header</code> in <code>HttpRequest</code>.</td>
</tr>
</tbody>
</table>

The following figure shows an example of accessing a real-time service with request parameter `images`.
Figure 7-18 Accessing a real-time service

4. Execute the following function to add the **X-Sdk-Date** header and **Authorization** header used for signature to the request:
   ```python
   sig.Sign(r)
   ```

5. Call the API and check the result.
   ```python
   resp = requests.request(method,url, headers, data, files)
   ```

---

Table 7-8 Request parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>Request method of the signed request object.</td>
</tr>
<tr>
<td>url</td>
<td>Request URL of the signed request object.</td>
</tr>
<tr>
<td>headers</td>
<td>Headers of the signed request object.</td>
</tr>
<tr>
<td>data</td>
<td>Body of the request object, which can only be in JSON format.</td>
</tr>
<tr>
<td>files</td>
<td>Request body in multipart/form-data format.</td>
</tr>
</tbody>
</table>

a. If the request body is in JSON format:
   ```python
   resp = requests.request(r.method, r.scheme + "://" + r.host + r.uri, headers=r.headers, data=r.body)
   print(resp.status_code, resp.reason)
   print(resp.content)
   ```

b. If the request body is input as an image:
7.2.6 Accessing a Real-Time Service (Application Authentication)

You can enable application authentication when deploying a real-time service. ModelArts registers an API that supports application authentication for the service. After this API is authorized to an application, you can call this API using the AppKey/AppSecret or AppCode of the application.

The process of application authentication for a real-time service is as follows:

- **Enabling Application Authentication**: Enable application authentication and create an application.
- **Managing Authorization of Real-Time Services**: Manage the created application, including viewing, resetting, or deleting the application, binding or unbinding real-time services for the application, and obtaining the AppKey/AppSecret or AppCode.
- **Application Authentication**: Authentication is required for calling an API that supports application authentication. Two authentication modes are provided. You can select either of them.
- **Preparing Environment (Python)**: After application authentication is complete, prepare the calling environment to call the API.
- **API Calling Example**: Use the authorized application information to call this API for prediction.

**Prerequisites**

- A ModelArts model in the **Normal** state is available.
- The account is not in arrears to ensure available resources for service running.

**Enabling Application Authentication**

When deploying a model as a real-time service, you can enable application authentication. You can also modify a deployed real-time service to support application authentication.

1. Log in to the ModelArts management console and choose **Service Deployment > Real-Time Services**.
2. Enable application authentication.
   - When deploying a model as a real-time service, configure the required parameters and enable application authentication on the **Deploy** page.
   - For a deployed real-time service, go to the **Real-Time Services** page, and click **Modify** in the **Operation** column of the service. On the service modification page that is displayed, enable application authentication.

**Figure 7-19** Enabling application authentication
3. Select an application for authorization from the drop-down list. If no application is available, create one as follows:

- Click **Create Application** on the right, enter the application name and description, and click **OK**. By default, the application name starts with `app_`. You can change the application name.

- On the **Real-Time Services** page, click **Authorize**. On the **Manage Authorization of Real-Time Services** page, click **Create Application**. For details, see **Managing Authorization of Real-Time Services**.

4. After enabling application authentication, authorize a service that supports application authentication to the application. Then, you can use the generated AppKey/AppSecret or AppCode to call the service's API that supports application authentication.

**Managing Authorization of Real-Time Services**

If you want to use application authentication, it is a good practice to create an application on the authorization management page before deploying a real-time service. On the **Real-Time Services** page, click **Authorize**. The **Manage Authorization of Real-Time Services** page is displayed. On this page, you can create and manage applications, including viewing, resetting, and deleting applications, unbinding real-time services from applications, and obtaining the AppKeys/AppSecrets or AppCodes.

**Figure 7-20** Managing authorization for real-time services

- **Creating an application**
  
  Click **Create Application**, enter the application name and description, and click **OK**. By default, the application name starts with `app_`. You can change the application name.

- **Viewing, resetting, or deleting an application**
  
  View, reset, or delete an application by clicking the corresponding icon in the **Operation** column of the application. After an application is created, the AppKey and AppSecret are automatically generated for application authentication.

- **Unbinding a service**
  
  In front of the target application name, click ✅ to view the real-time services bound to the application. Click **Unbind** in the **Operation** column to cancel the binding. Then, this API cannot be called.
• Obtaining the AppKey/AppSecret or AppCode

Application authentication is required for API calling. The AppKey and AppSecret are automatically generated during application creation. The AppCode is automatically generated by adding an application.

![Figure 7-21 Adding the AppCode](image)

Application Authentication

When a real-time service that supports application authentication is in the Running state, the service’s API can be called. Before calling the API, perform application authentication.

Requests for calling an API that supports application authentication can be authenticated using either AppKey/AppSecret-based authentication or AppCode-based authentication. AppKey/AppSecret-based authentication is recommended because it is more secure than AppCode-based authentication.

- AppCode-based authentication: Requests are authenticated using AppCodes.
- AppKey/AppSecret-based authentication: The AppKey (application access key ID) and AppSecret (application secret access key) are used to encrypt a request, identify the sender, and prevent the request from being modified.

You can obtain the AppKey/AppSecret and AppCode from the Usage Guides tab page on the service details page (see Figure 7-22) or from the real-time service authorization management page (see Figure 7-20).

![Figure 7-22 Obtaining the API address](image)

AppCode-based authentication

When application authentication is used and the simple authentication mode is enabled, you can use the AppKey/AppSecret for signature and verification, or AppCode for simple authentication of API requests. By default, AppCode-based authentication is enabled for ModelArts.

In AppCode-based authentication, the X-Apig-AppCode parameter (value: AppCode) is added to the HTTP request header when an API is called. The request
content does not need to be signed. The API gateway only verifies the AppCode and does not verify the request signature, achieving quick response. The sample code is as follows:

```plaintext
GET https://iam.cn-north-1.myhuaweicloud.com/v3/auth/projects
Content-Type: application/json
X-Apig-AppCode: ABCDEFG....
```

**NOTE**

If an application does not support AppCodes, AppKeys can be used for simple authentication. That is, when an API is called, the `apikey` parameter (value: `AppKey`) can be added to the HTTP request header to accelerate authentication.

**AppKey/AppSecret-based authentication**

In AppKey/AppSecret-based authentication, the AppKey and AppSecret of an application are used for signature authentication.

- **AppKey**: access key ID of the application, which is a unique identifier used together with a secret access key to sign requests cryptographically.
- **AppSecret**: secret access key used together with an AppKey to sign requests. The AppKey and AppSecret can be used together to identify a request sender to prevent the request from being modified.

When using AppKey/AppSecret-based authentication, use a dedicated signing SDK to sign requests.

**Preparing Environment (Python)**

After application authentication is complete, you can prepare the calling environment to call the API. This section uses Python as an example to describe how to use the SDK. For details about how to download and use the SDK in other languages, see the API Gateway Developer Guide.

1. Obtain the API address, request method, AppKey, and AppSecret. For details, see Application Authentication.
2. Obtain the Python installation package (version 2.7.9 or 3.X) from the Python official website and install it. After Python is installed, you can run the `pip install requests` command to install the `requests` library using the Python package manager pip.
   
   **NOTE**

   If a certificate error occurs when you use pip to install the requests library, download and execute the file using Python to upgrade pip. Then run the preceding command to install the library.

3. Obtain IntelliJ IDEA from the IntelliJ IDEA official website and install it. Install the Python plug-in on IntelliJ IDEA. See Figure 7-23.
Figure 7-23 Installing the Python plug-in

4. Download the ApiGateway-python-sdk.zip. The following table describes the directory structure of the decompressed package.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apig_sdk_<em>init</em>_.py</td>
<td>SDK code</td>
</tr>
<tr>
<td>apig_sdk\signer.py</td>
<td>Sample code</td>
</tr>
<tr>
<td>main.py</td>
<td>Sample code for the backend signature</td>
</tr>
<tr>
<td>backend_signature.py</td>
<td></td>
</tr>
<tr>
<td>licenses\license-requests</td>
<td>Third-party library license file</td>
</tr>
</tbody>
</table>

5. Create a project.
b. Click Next. The following page is displayed. Click ..., select the path for the decompressed SDK package, and click Finish.

Figure 7-25 Selecting the SDK path after decompression

6. After the project is created, the directory structure is shown in the following figure. main.py is sample code. Modify the parameters to suit your requirements before use. For details about the sample code, see API Calling Example.
Figure 7-26 Directory structure of a new project

```
python D:\java\python
  \ idea
  \ ApiGateway-python-sdk-3.0.0
    \ api_sdk
      \ __init__.py
      \ signer.py
  \ licenses
  \ main.py
  \ test_signer.py
  \ python.ini
```

**API Calling Example**

1. Add `apig_sdk` to the project.
   ```
   from apig_sdk import signer
   import requests
   ```

2. Generate a new signer and enter the AppKey and AppSecret. For details about how to obtain the AppKey and AppSecret, see **Application Authentication**.
   ```
   sig = signer.Signer()
   sig.Key = "4f5f626b-073f-402f-a1e0-e52171c6100c"
   sig.Secret = "******"
   ```

3. Generate a request object and specify the method, request URI, header, and body.
   ```
   r = signer.HttpRequest(method, uri, header, body)
   ```

**Table 7-10** HttpRequest parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sub-Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>N/A</td>
<td>Yes</td>
<td>The value can be <strong>GET, POST, PUT, or DELETE</strong>.</td>
</tr>
<tr>
<td>uri</td>
<td>N/A</td>
<td>Yes</td>
<td>Enter the API URI of the real-time service. For details about how to obtain the URI, see <strong>Application Authentication</strong>.</td>
</tr>
<tr>
<td>header</td>
<td>x-stage</td>
<td>Yes</td>
<td>API release environment, which can only be <strong>RELEASE</strong>.</td>
</tr>
<tr>
<td></td>
<td>Content-Type</td>
<td>No</td>
<td>Content type, which can only be <strong>application/json</strong>. For details about the multipart/form-data request body, see <strong>Table 7-11</strong>.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Sub-Parameter</td>
<td>Mandatory</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>x-sdk-content-sha256</td>
<td></td>
<td>No</td>
<td>Signature mode. This parameter can be set to UNSIGNED-PAYLOAD, indicating that signature authentication is not performed on the request body. This parameter is mandatory when the body is input as a file.</td>
</tr>
</tbody>
</table>

| body | N/A | No | The JSON format is supported, for example, "{"xxx":"xxx"}". |

a. If the request body is in JSON format:

```python
r = signer.HttpRequest("POST", "https://1684994b180244de9d141c00d3e52c73.apig.exampleRegion.huaweicloudapis.com/v1/infers/exampleServiceId", {"x-stage": "RELEASE","Content-Type":"application/json"},{"{"xxx"":"xxx"}})
```

b. If the request body is input as an image, construct the request body in multipart/form-data format.

Request body format: files = {"Request parameter": ("Load path", File content, "File type", Request header)}

### Table 7-11 files parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request parameter</td>
<td>Enter the parameter name of the real-time service.</td>
</tr>
<tr>
<td>Load path</td>
<td>Path in which the file is stored.</td>
</tr>
<tr>
<td>File content</td>
<td>Content of the file to be uploaded.</td>
</tr>
</tbody>
</table>
| File type | Type of the file to be uploaded, which can be one of the following options:  
  ● txt: text/plain  
  ● jpg/jpeg: image/jpeg  
  ● png: image/png |
| Request header | You are advised to set this parameter to {}. The request header is specified by header in HttpRequest. |

If the input parameter of the real-time service you access is images and the parameter type is file, the request example is as follows:
Figure 7-27 Accessing a real-time service

![API Address](https://4c10d8f3c82c4aa98656c0784f2f11907.apig.cn-north-4.huaweicloudapis.com/v1/infers/76c41384-23ab-45f9-a66e-892e7bc2be53, "x-stage": "RELEASE", "x-sdk-content-sha256": "UNSGNDED-PAYLOAD")

```python
files = {"images": ("flower.png", open("flower.png", "rb"), "image/png", {})}
```

4. Execute the following function to add the **X-Sdk-Date** header and **Authorization** header used for signature to the request:

```python
sig.Sign(r)
```

5. Call the API and check the result.

```python
resp = requests.request(method,url, headers, data, files)
```

### Table 7-12 Request parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>Request method of the signed request object.</td>
</tr>
<tr>
<td>url</td>
<td>Request URL of the signed request object.</td>
</tr>
<tr>
<td>headers</td>
<td>Headers of the signed request object.</td>
</tr>
<tr>
<td>data</td>
<td>Body of the request object, which can only be in JSON format.</td>
</tr>
<tr>
<td>files</td>
<td>Request body in multipart/form-data format.</td>
</tr>
</tbody>
</table>

a. If the request body is in JSON format:

```python
resp = requests.request(r.method, r.scheme + "://" + r.host + r.uri, headers=r.headers, data=r.body)
print(resp.status_code, resp.reason)
print(resp.content)
```

b. If the request body is input as an image:
7.2.7 Integrating a Real-Time Service

For a real-time service API that has been commissioned, you can integrate it into the production environment.

Prerequisites

The real-time service is running. Otherwise, applications in the production environment will be unavailable.

Integration Mode

ModelArts real-time services provide standard RESTful APIs, which can be accessed using HTTPS. ModelArts provides SDKs for calling real-time service APIs. For details about how to call the SDKs, see Scenario 1: Perform an inference test using the predictor deployed in Deploying a Real-Time Service in SDK Reference.

In addition, you can use common development tools and languages to call the APIs. You can search for and obtain the guides for calling standard RESTful APIs on the Internet.

7.2.8 Collecting Data

ModelArts provides an auto hard example identification function for you to filter hard example data from inference data inputted to an existing model based on built-in rules. This improves model precision, and effectively reduces labeling manpower required upon a model update. This function helps mine data that benefits model precision improvement as much as possible. You only need to confirm and label useful data and add it to a training dataset. Then, a new model with higher precision can be obtained after training.

You can call a URL or use the console to input data for a model deployed as a real-time service. Then, use the data collection function to collect data or filter out hard examples, and output them to a dataset for future model training.

For real-time services, data collection involves the following scenarios, as shown in Figure 7-28.

- **Data Collection**: Enable a data collection task to collect and store data generated when a real-time service is invoked based on the configured rules.
- **Synchronizing Data to a Dataset**: Synchronize the collected data to a dataset for unified management and application.
- **Data Collection and Hard Example Filtering**: Enable the hard example filtering function in addition to the data collection task to filter hard examples from the collected data using built-in algorithms. Finally, store hard examples and the collected data in a corresponding dataset for retraining.
- **Hard Example Feedback**: When calling a real-time service for prediction, report inaccurately predicted image data as hard examples and store them to the corresponding dataset.
Prerequisites

- A trained model has been deployed as a real-time service, and the real-time service is in the **Running** status.
- The type of the real-time service is proper. Data collection and hard example filtering are available for the object detection and image classification types only.

Data Collection

You can enable a data collection task when deploying a model as a real-time service or on the service details page after the real-time service is deployed. If only a data collection task is enabled, data generated during service invoking is merely collected and stored to OBS. If you want to filter hard examples, follow instructions in **Data Collection and Hard Example Filtering**. If you want to synchronize the collected data to a dataset and do not require hard example filtering, follow instructions in **Synchronizing Data to a Dataset**.

1. Log in to the ModelArts management console and choose Service Deployment > Real-Time Services.
2. Enable a data collection task.
   - When deploying a model as a real-time service, enable Data Collection on the Deploy page.
After a real-time service is deployed, click the service name to go to the service details page. In the Sample area, enable a data collection task.

**Figure 7-30** Enabling the data collection function on the details page

![Image](image1.jpg)  

3. Set parameters related the data collection task. **Table 7-13** describes the parameters.

**Table 7-13** Data collection parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Collection Rule</td>
<td>Possible values are <strong>Full collection</strong> and <strong>By confidence score</strong>. Currently, only <strong>Full collection</strong> is available.</td>
</tr>
<tr>
<td>Sample Output Path</td>
<td>Path for storing collected data. Only OBS paths are supported. Select an existing path or create an OBS path.</td>
</tr>
</tbody>
</table>
| Retention Period      | Possible values are **1 day**, **1 week**, **Permanent**, and **Custom**.  
  - **1 day**: Only data generated during service running within one day is collected.  
  - **1 week**: Only data generated during service running within one week is collected.  
  - **Permanent**: All data after a service is started is collected.  
  - **Custom**: The parameter can only be set to **X** days, indicating that data generated during service running within **X** days is collected. |

**Figure 7-31** Data collection configuration

![Image](image2.jpg)  

After data collection is enabled, the uploaded data is collected to the corresponding OBS path based on the configured rules when the service is invoked for prediction either through the console or URL APIs.
Synchronizing Data to a Dataset

For real-time services with data collection enabled, the collected data can be synchronized to a dataset. The synchronization operation merely stores the collected data to the dataset without hard examples filtered. You can select an existing dataset or create a dataset to store data.

1. Enable a data collection task. For details, see Data Collection.
   
   If no data is collected in the data collection task because you do not invoke an API to implement prediction, synchronizing data to the dataset cannot be implemented.

2. Click the service name to go to the service details page. In the Synchronize Data area, click Synchronize Data to Dataset.

   Figure 7-32 Synchronizing data to a dataset

3. In the displayed dialog box, select a labeling type and a dataset, and click OK to synchronize the collected data to the dataset. The synchronized data will be displayed on the Unlabeled tab page of the dataset.

   Data to be synchronized is the data collected by the system in a data collection task based on the configured rules. If no data is collected, data synchronization cannot be implemented.

   Figure 7-33 Synchronizing data to a dataset

Data Collection and Hard Example Filtering

If you only enable a data collection task, hard examples cannot be automatically identified. To filter hard examples from the collected data and store filtering results to a dataset, you need to enable both data collection and hard example filtering tasks.
The hard example filtering function has requirements on the prediction output format, which varies depending on the model source.

- For models trained by ExeML, you do not need to modify them. The prediction output format of ExeML is built in the system, which meets the requirements of hard example filtering.
- For models trained by built-in algorithms, you do not need to modify them. The prediction output format of built-in algorithms is built in the system, which meets the requirements of hard example filtering.
- For models you trained, ensure that the output format in the inference code meets the requirements. The requirements for object detection are different from that for image classification. For details, see Prediction Output Format Requirements. For example, if you use a frequently-used framework or a custom image to train a model, the prediction output format must meet the requirements of the corresponding type when you import the model and compile the inference code.

1. Enable a data collection task. For details, see Data Collection.

   **NOTE**
   
   A data collection task must be enabled first before you enable hard example filtering. If a data collection task has been enabled for a real-time service and data is still stored in the OBS path, you can enable only the hard example filtering function. In this case, hard examples are filtered from only the data stored in the OBS path.

2. Enable a hard example filtering task on the same page for configuring the data collection task. For details about the parameters, see Table 7-14.

<table>
<thead>
<tr>
<th>Table 7-14 Hard example filtering parameters</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Type</td>
<td>Model application type. Currently, only <strong>image classification</strong> and <strong>Object detection</strong> are supported.</td>
</tr>
<tr>
<td>Training Dataset</td>
<td>A model is trained based on a dataset and can be deployed as a real-time service. When filtering hard examples, you can import the dataset corresponding to the real-time service to find data problems underlying the model. The model training and deployment process is as follows: <em>Input training scripts and a dataset.</em> &gt; <em>Train the dataset to obtain a model.</em> &gt; <em>Deploy the model as a real-time service.</em> This parameter is optional. You are advised to import the dataset to improve training precision. If your dataset is not managed on ModelArts, see Creating a Dataset.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Filtering Policy          | The options are **By duration** and **By sample quantity**.  
  - **By duration**: Filter data that is stored in an OBS path and is not filtered by duration. Possible values are 1 hour, 1 day, 2 days, and Custom. The value of Custom can only be XX hours.  
  **NOTE**  
  If you filter hard examples by duration, the duration must be less than the value of Retention Period specified for data collection. For example, if Retention Period is set to 1 day for data collection, **By duration** must be set to a value equal to or less than 1 day for Filtering Policy. If the duration is greater than the retention period, the system filters only the data within the retention period.  
  - **By sample quantity**: When the collected data reaches the sample quantity, the system performs once hard example filtering. Possible values are 100, 500, 1000, and Custom. If a sample quantity is less than the value specified in the filtering policy within the data collection period, hard example filtering will not be enabled. For example, if Retention Period is set to 1 day for data collection and only 100 images generated within 1 day, hard example filtering will not be enabled when **By sample quantity** is set to 500 in Filtering Policy. OBS deletes data that has been stored for more than one day based on the retention period. In this case, the number of samples does not increase and the filtering criteria cannot be met. Therefore, when setting a filtering policy, evaluate the service invoking volume, and set sample quantity based on the site requirements. |
| Hard Example Output       | Save the filtered hard example data to a dataset. You can select an existing dataset or create a new dataset.  
  A dataset type must match a model type. For example, if the model type is image classification, the dataset to which hard example data is outputted must be image classification. |
3. After data collection and hard example filtering tasks are configured, the system collects data and filters hard examples based on the configured rules. You can view the task status on the Filter tab page of the real-time service. After the task is complete, its status changes to Dataset imported. You can click the dataset link to quickly access the corresponding dataset. The collected data is stored on the Unlabeled tab page. The filtered hard examples are stored on the To Be Confirmed tab page of the dataset.

Figure 7-34 Enabling hard example filtering

Figure 7-35 Task status

Figure 7-36 Hard example filtering result
Hard Example Feedback

On the ModelArts management console, if the prediction result of a real-time service is inaccurate, you can directly report it as a hard example to the corresponding dataset on the **Prediction** tab page.

1. Log in to the ModelArts management console and choose **Service Deployment > Real-Time Services**. Click the service name to go to the service details page.

2. Click the **Prediction** tab, upload the image for prediction, and click **Predict**.

3. If the prediction result is inaccurate, click **Feed Back**.

4. In the displayed dialog box, select a labeling type and a dataset, and click **OK** to report the hard example to the dataset. The hard example will be displayed on the **Unlabeled** tab page of the dataset. This helps improve model training precision.
Prediction Output Format Requirements

For a custom model, `infer_output` in the inference code, that is, the JSON format returned by the inference engine, must be the same as that in the following example.

- **Object detection**
  
The prediction output format is as follows:

```json
{
    "detection_classes": [
        "<label-name-1>",
        "<label-name-2>"
    ],
    "detection_boxes": [
        [<y_min>,
        [<x_min>,
        [<y_max>,
        [<x_max>]
        ],
        [<y_min>,
        [<x_min>,
        [<y_max>,
        [<x_max>]
        ]
    ],
    "detection_scores": [
        [<label-1-score>],
        [<label-2-score>]
    ]
}
```

<table>
<thead>
<tr>
<th>Table 7-15 Parameters in the prediction result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>detection_classes</td>
</tr>
<tr>
<td>detection_boxes</td>
</tr>
<tr>
<td>detection_scores</td>
</tr>
</tbody>
</table>
The prediction output format is as follows:

```json
{
    "predicted_label": "<label-name-1>",
    "scores": [
        ["<label-name-1>",
         "<label-1-score>"],
        ["<label-name-2>",
         "<label-2-score>"]
    ]
}
```

### Table 7-16 Parameters in the prediction result

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>predict_label</td>
<td>Image prediction label</td>
</tr>
<tr>
<td>scores</td>
<td>Prediction confidence of top 5 labels</td>
</tr>
</tbody>
</table>

### 7.3 Batch Services

#### 7.3.1 Deploying a Model as a Batch Service

After a model is prepared, you can deploy it as a batch service. The **Service Deployment > Batch Services** page lists all batch services. You can enter a service name in the search box in the upper right corner and click **Search** to query the service.
Prerequisites

- Data has been prepared. Specifically, you have created a model in the Normal state in ModelArts.
- Data to be batch processed is ready and has been upload to an OBS directory.
- At least one empty folder has been created on OBS for storing the output.

Background

- Currently, batch services are limited-time free. The running batch services are not billed.
- A maximum of 1,000 batch services can be created.
- Based on the input request (JSON or other file) defined by the model, different parameter are entered. If the model input is a JSON file, a configuration file is required to generate a mapping file. If the model input is other file, no mapping file is required.

Procedure

1. Log in to the ModelArts management console. In the left navigation pane, choose Service Deployment > Batch Services. By default, the system switches to the Batch Services page.
2. In the batch service list, click Deploy in the upper left corner. The Deploy page is displayed.
3. Set parameters for a batch service.
   a. Set the basic information, including Name and Description. The name is generated by default, for example, service-bc0d. You can specify Name and Description according to actual requirements.
   b. Set other parameters, including the resource pool and model configurations. For details, see Table 7-17.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model and Version</td>
<td>Select the model and version that are in the Normal status.</td>
</tr>
<tr>
<td>Input Path</td>
<td>Select the OBS directory where the data is to be uploaded. Select a folder or a .manifest file. For details about the specifications of the .manifest file, see Manifest File Specifications. <strong>NOTE</strong> If the input data is an image, it is recommended that the size of a single image be less than 10 MB.</td>
</tr>
<tr>
<td>Request Path</td>
<td>API URI of a batch service.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Mapping Relationship</td>
<td>If the model input is in JSON format, the system automatically generates the mapping based on the configuration file corresponding to the model. If the model input is other file, mapping is not required. Automatically generated mapping file. Enter the field index corresponding to each parameter in the CSV file. The index starts from 0. Mapping rule: The mapping rule comes from the input parameter (request) in the model configuration file <code>config.json</code>. When <code>type</code> is set to <code>string/number/integer/boolean</code>, you need to configure this parameter, that is, the index parameter. For details about the mapping rule, see <strong>Example Mapping</strong>. The index must be a positive integer starting from 0. If the value of index does not comply with the rule, this parameter is ignored in the request. After the mapping rule is configured, the corresponding CSV data must be separated by commas (,).</td>
</tr>
<tr>
<td>Output Path</td>
<td>Select the path for saving the batch prediction result. You can select the empty folder that you create.</td>
</tr>
<tr>
<td>Specifications</td>
<td>The system provides available compute resources matching your model. Select an available resource from the drop-down list. For example, if the model comes from an ExeML project, the compute resources are automatically associated with the ExeML specifications for use.</td>
</tr>
<tr>
<td>Compute Nodes</td>
<td>Set the number of instances for the current model version. If you set <code>Instances</code> to 1, the standalone computing mode is used. If you set <code>Instances</code> to a value greater than 1, the distributed computing mode is used. Select a computing mode based on the actual requirements.</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Set environment variables and inject them to the container instance.</td>
</tr>
</tbody>
</table>

4. After setting the parameters, deploy the model as a batch service as prompted. Generally, service deployment jobs run for a period of time, which may be several minutes or tens of minutes depending on the amount of your selected data and resources.

**NOTE**

After a batch service is deployed, it is started immediately. During the running, you will be charged based on your selected resources.
You can go to the batch service list to view the basic information about the batch service. In the batch service list, after the status of the newly deployed service changes from Deploying to Running, the service is deployed successfully.

Manifest File Specifications

Batch services of the inference platform support the manifest file. The manifest file describes the input and output of data.

Example input manifest file

- **File name:** test.manifest
- **File content:**
  
  ```javascript
  {"source": "<obs path>/test/data/1.jpg"}
  
  {"source": "https://infers-data.obs.cn-north-1.myhuaweicloud.com:443/xgboosterdata/data.csv?
AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttJ5QZrReDhz1uDzwwe
8GpyY%3D&-obs-security-token=qGqzsb3VoGNoaWShixvY8V9a1SnmxmGoHYmB1SArYMyqntQ-
ZatxSxHw168kKlA55fevLDLMN2WxzhB6Z5Q-3Hcozm9i5SwQOVBwvm42yTb_m8sgfl6isU7T3Cnlo9jm
vDGq79v96C7C117eyfIsUqcf8N0ykc0frATtJQyZFDu_HqVvK-
GUnVcTDvDlCVC37yPcmmznJJ1nYU09kAwCYGceZxCo0ePu4PHMsb5bYV9gWmN9AUZID1n5frL4vo
BwpqYn6t6AgYw4ySa6hP2hCAoQ-955SurniJ434QylymoeKfTHVMKoEzSze-
JxOveOvC65GehEiaz48s6H81uiHzi21zocNB_hpPfus2Y6K pkEikMv6kWmrr-
ZBXwusUJUDmSXYk-3cYj9g-
h10bW3s81VasoFCW8GdWsd74it7L_5-7UhoelyPByO_Rewkur2F0JsuMgIraPyglZxXm_jfdLFxobYtz
Zhu6u4yWxg96oX7OkcwykTOYH0NPhPrt5MYGYweOXXx5s6d5w2rdz7YpOQHyzTzkK5ClzT7FIWnapi6L
7zdhs18sf6f77E5p4kkgkqextSF_xlymVMWzr8P6z2e6h6N7LgqtmBo98WecfrnD2Thn76h_1n2H8h1-
H0tj
zkKvqngaimYhBbMRplftvtW86GwvP9c27FOn39Be9zHsf1_8pHej0yMlyNqZ48lfFQ5vWT_yFV3HJHM-
71L2Z0_hdaHtflJ69cTFH5EOxt7DGAeMIES107U3w%3D%3D")
  ```

- **File requirements:**
  
  a. The file name extension must be .manifest.
  
  b. The file content is in JSON format. Each row describes a piece of input data, which must be accurate to a file instead of a folder.
  
  c. A source field must be defined for the JSON content. The field value is the OBS URL of the file in any of the following formats:
  
  i. `<obs path>/<{{Bucket name}}>/<{{Object name}}>`: applicable to accessing OBS data of the user
  
  ii. Shared link generated by OBS, including signature information. It is applicable to accessing OBS data of other users.

Example output manifest file

If you use an input manifest file, the output directory will contain an output manifest file.

- **Assume that the output path is //test-bucket/test/.** The result is stored in the following path:
  
  **OBS bucket/directory name**

  ```
  test
  
  test
  
  infer-result-0
  
  infer-result
  
  1.jpg_result.txt
  
  2.jpg_result.txt
  ```

- **Content of the infer-result-0.manifest file:**
  
  ```javascript
  {"source": "<obs path>/obs-data-bucket-test/data/1.jpg", "inference-loc": "<obs path>/test-bucket/test/infer-result/1.jpg_result.txt"}
  
  {"source": "https://infers-data.obs.cn-north-1.myhuaweicloud.com:443/xgboosterdata/1.jpg?
AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttJ5QZrReDhz1uDzwwe
8GpyY%3D&-obs-security-token=qGqzsb3VoGNoaWShixvY8V9a1SnmxmGoHYmB1SArYMyqntQ-
ZatxSxHw168kKlA55fevLDLMN2WxzhB6Z5Q-3Hcozm9i5SwQOVBwvm42yTb_m8sgfl6isU7T3Cnlo9jm
vDGq79v96C7C117eyfIsUqcf8N0ykc0frATtJQyZFDu_HqVvK-
GUnVcTDvDlCVC37yPcmmznJJ1nYU09kAwCYGceZxCo0ePu4PHMsb5bYV9gWmN9AUZID1n5frL4vo
BwpqYn6t6AgYw4ySa6hP2hCAoQ-955SurniJ434QylymoeKfTHVMKoEzSze-
JxOveOvC65GehEiaz48s6H81uiHzi21zocNB_hpPfus2Y6K pkEikMv6kWmrr-
ZBXwusUJUDmSXYk-3cYj9g-
h10bW3s81VasoFCW8GdWsd74it7L_5-7UhoelyPByO_Rewkur2F0JsuMgIraPyglZxXm_jfdLFxobYtz
Zhu6u4yWxg96oX7OkcwykTOYH0NPhPrt5MYGYweOXXx5s6d5w2rdz7YpOQHyzTzkK5ClzT7FIWnapi6L
7zdhs18sf6f77E5p4kkgkqextSF_xlymVMWzr8P6z2e6h6N7LgqtmBo98WecfrnD2Thn76h_1n2H8h1-
H0tj
zkKvqngaimYhBbMRplftvtW86GwvP9c27FOn39Be9zHsf1_8pHej0yMlyNqZ48lfFQ5vWT_yFV3HJHM-
71L2Z0_hdaHtflJ69cTFH5EOxt7DGAeMIES107U3w%3D%3D")
  ```
● File format:
  a. The file name is `infer-result-{{index}}.manifest`, where `index` is the instance ID. Each running instance of a batch service generates a manifest file.
  b. The `infer-result` directory is created in the manifest directory to store the result.
  c. The file content is in JSON format. Each row describes the output result of a piece of input data.
  d. The content contains two fields:
     i. `source`: input data description, which is the same as that of the input manifest file
     ii. `inference-loc`: output result path in the format of `<obs path>/{Bucket name}/{Object name}`

**Example Mapping**

The following example shows the relationship between the configuration file, mapping rule, CSV data, and inference request.

Assume that the `apis` parameter in the configuration file used by your model is as follows:

```json
[
  {
    "protocol": "http",
    "method": "post",
    "url": "/",
    "request": {
      "type": "object",
      "properties": {
        "data": {
          "type": "object",
          "properties": {
            "req_data": {
              "type": "array",
              "items": {
                "type": "object",
                "properties": {
                  "input_1": {
                    "type": "number"
                  },
                  "input_2": {
                    "type": "number"
                  },
                  "input_3": {
                    "type": "number"
                  }
                }
              }
            }
          }
        }
      }
    }
  }
]```

Copyright © Huawei Technologies Co., Ltd.
At this point, the corresponding mapping relationship is shown below. The ModelArts management console automatically resolves the mapping relationship from the configuration file. When calling a ModelArts API, you need to write the mapping relationship yourself according to the rule.

```json
{
  "type": "object",
  "properties": {
    "data": {
      "type": "object",
      "properties": {
        "req_data": {
          "type": "array",
          "items": [
            {
              "type": "object",
              "properties": {
                "input_1": {
                  "type": "number",
                  "index": 0
                },
                "input_2": {
                  "type": "number",
                  "index": 1
                },
                "input_3": {
                  "type": "number",
                  "index": 2
                },
                "input_4": {
                  "type": "number",
                  "index": 3
                }
              }
            }
          ]
        }
      }
    }
  }
}
```

The data for inference, that is, the CSV data, is in the following format. The data must be separated by commas (,).

```
5.1,3.5,1.4,0.2
4.9,3.0,1.4,0.2
4.7,3.2,1.3,0.2
```

Depending on the defined mapping relationship, the inference request is shown below. The format is similar to the format used by the real-time service.

```json
{
  "data": {
  
  }
}
```
7.3.2 Viewing the Batch Service Prediction Result

When deploying a batch service, you can select the location of the output data directory. You can view the running result of the batch service that is in the Running completed status.

Procedure

1. Log in to the ModelArts management console and choose Service Deployment > Batch Services.
2. Click the name of the target service in the Running completed status. The service details page is displayed.
   - You can view the service name, status, ID, input path, output path, and description.
   - You can click \( \text{Description} \) in the Description area to edit the description.

![Figure 7-40 Service details page](image)

3. Obtain the detailed OBS path next to Output Path, switch to the path and obtain the batch service prediction result.
   - If images are entered, a result file is generated for each image in the Image name__result.txt format, for example, IMG_20180919_115016.jpg_result.txt.
   - If audio files are entered, a result file is generated for each audio file in the Audio file name__result.txt format, for example, 1-36929-A-47.wav_result.txt.
   - If table data is entered, the result file is generated in the Table name__result.txt format, for example, train.csv_result.txt.

7.3.3 Collecting Data

ModelArts provides an auto hard example identification function for you to filter hard example data from inference data inputted to an existing model based on built-in rules. This improves model precision, and effectively reduces labeling...
manpower required upon a model update. This function helps mine data that benefits model precision improvement as much as possible. You only need to confirm and label useful data and add it to a training dataset. Then, a new model with higher precision can be obtained after training.

For models deployed as batch services, data generated during service invoking is stored in an OBS directory by default. ModelArts can automatically filter hard examples from existing data based on configured rules, and output them to a dataset for future model training.

For batch services, data synchronization and hard example filtering involve the following scenarios, as shown in Figure 7-41.

- **Synchronizing Data to a Dataset**: Synchronize the input data of a batch service to a dataset for unified management and application.
- **Hard Example Filtering**: Enable the hard example filtering function to filter hard examples from the input data of a batch service using built-in algorithms. Finally, store hard examples in a corresponding dataset for retraining.

**Figure 7-41** Data collection for batch services

**Synchronizing Data to a Dataset**

The input data of batch services can be synchronized to a dataset. The synchronization operation merely stores data to the dataset without hard examples filtered. You can select an existing dataset or create a dataset to store data.

1. Log in to the ModelArts management console and choose **Service Deployment > Batch Services**.
2. Click the service name to go to the service details page, and click the **Sample** tab. Alternatively, choose **More > Sample Collection** in the **Operation** column in the service list.

![Figure 7-42 Accessing the data collection page from the Batch Services page](image)

3. On the **Sample** tab page, click **Synchronize Data to Dataset**.
4. In the displayed dialog box, select a labeling type and a dataset, and click **OK** to synchronize data to the dataset. The synchronized data will be displayed on the **Unlabeled** tab page of the dataset.

   If there is no input data of a batch service, data synchronization cannot be implemented.

![Figure 7-43 Synchronizing data to the dataset for a batch service](image)

**Hard Example Filtering**

To filter hard examples from batch service data and store filtering results to a dataset, you need to enable hard example filtering tasks.

**NOTE**

If batch service running is complete and hard example filtering is disabled, a hard example filtering task will not be executed. After configuring a hard example filtering task, you need to restart the batch service to execute the task.

1. Log in to the ModelArts management console and choose **Service Deployment > Batch Services**.
2. Enable a hard example filtering task
   - When deploying a model as a batch service, enable **Hard Example Filtering** on the **Deploy** page.
After a batch service is deployed, click the service name to go to the service details page. Click the edit icon next to **Hard Example Filtering** to enable a hard example filtering task.

**Figure 7-45** Enabling the **Hard Example Filtering** function on the details page

3. Set the parameters related to hard example filtering. For details, see **Table 7-18**. For batch services, hard examples are filtered from all data and no filtering policy is required, which is different from real-time services.

**Table 7-18** Hard example filtering parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model Type</strong></td>
<td>Model application type. Currently, only <strong>image classification</strong> and <strong>object detection</strong> are supported.</td>
</tr>
<tr>
<td><strong>Existing Training Dataset Path</strong></td>
<td>A model is trained based on a dataset and can be deployed as a batch service. When filtering hard examples, you can import the manifest file of the dataset corresponding to the batch service to find data problems underlying the model. The model training and deployment process is as follows: <strong>Input training scripts and a dataset. &gt; Train the dataset to obtain a model. &gt; Deploy the model as a batch service.</strong> This parameter is optional. You are advised to import the dataset to improve training precision. Currently, only the manifest file of the dataset can be imported. If a dataset is managed on ModelArts, <strong>publish the dataset</strong> to obtain its manifest file. If your dataset is not managed on ModelArts, import the manifest file by referring to <strong>Specifications for Importing the Manifest File</strong>.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Hard Example Output</td>
<td>Save the filtered hard example data to a dataset. You can select an existing dataset or create a new dataset. A dataset type must match a model type. For example, if the model type is image classification, the dataset to which hard example data is outputted must be image classification.</td>
</tr>
</tbody>
</table>

**Figure 7-46** Enabling hard example filtering

Filter

![Filtering Policy](image)

- Model Type
- Existing Training Dataset Path
- Hard Example Output

After the hard example filtering task is configured and executed, view the task status on the **Sample** tab page of the batch service. After the task is complete, its status changes to **Dataset imported**. You can click the dataset link to quickly access the corresponding dataset. The filtered hard examples will be displayed on the **To Be Confirmed** tab page.

**Figure 7-47** Status of a data collection task
7.4 Modifying a Service

For a deployed service, you can modify its basic information to match service changes. You can modify the basic information about a service in either of the following ways:

- **Method 1: Modify Service Information on the Service Management Page**
- **Method 2: Modify Service Information on the Service Details Page**

**Prerequisites**

A service has been deployed.

**Method 1: Modify Service Information on the Service Management Page**

1. Log in to the ModelArts management console and choose Service Deployment from the left navigation pane. Go to the service management page of the target service.
2. In the service list, click Modify in the Operation column of the target service, modify basic service information, and click OK.
   - For details about the real-time service parameters, see Deploying a Model as a Real-Time Service.
   - For details about the batch service parameters, see Deploying a Model as a Batch Service.

**NOTE**

Services in the Deploying status cannot be modified.

**Method 2: Modify Service Information on the Service Details Page**

1. Log in to the ModelArts management console and choose Service Deployment from the left navigation pane. Go to the service management page of the target service.
2. Click the name of the target service. The service details page is displayed.

3. Click **Modify** in the upper right corner of the page, modify the service details, and click **OK**.
   - For details about the real-time service parameters, see **Deploying a Model as a Real-Time Service**.
   - For details about the batch service parameters, see **Deploying a Model as a Batch Service**.

### 7.5 Starting or Stopping a Service

#### Starting a Service

You can start services in the **Successful**, **Abnormal**, or **Stopped** status. Services in the **Deploying** status cannot be started. A service is billed when it is started and in the **Running** state. You can start a service in either of the following ways:

1. Log in to the ModelArts management console and choose **Service Deployment** from the left navigation pane. Go to the service management page of the target service. Click **Start** in the **Operation** column to start the target service.

2. Log in to the ModelArts management console and choose **Service Deployment** from the left navigation pane. Go to the service management page of the target service. Click the name of the target service. The service details page is displayed. Click **Start** in the upper right corner of the page to start the service.

#### Stopping a Service

You can stop services in the **Running** or **Alarm** status. Services in the **Deploying** status cannot be stopped. After a service is stopped, ModelArts stops charging. You can stop a service in either of the following ways:

1. Log in to the ModelArts management console and choose **Service Deployment** from the left navigation pane. Go to the service management page of the target service. Click **Stop** in the **Operation** column to stop the target service.

2. Log in to the ModelArts management console and choose **Service Deployment** from the left navigation pane. Go to the service management page of the target service. Click the name of the target service. The service details page is displayed. Click **Stop** in the upper right corner of the page to stop the service.

### 7.6 Deleting a Service

If a service is no longer in use, you can delete it to release resources.

1. Log in to the ModelArts management console and choose **Service Deployment** from the left navigation pane. Go to the service management page of the target service.
a. For a real-time service, choose More > Delete in the Operation column to delete it.

b. For a batch service, click Delete in the Operation column to delete it.

NOTE

A deleted service cannot be recovered. Exercise caution when performing this operation.
ModelArts Resource Pools

When using ModelArts to implement AI Development Lifecycle, you can use two different resource pools to train and deploy models.

- **Public Resource Pool**: provides public large-scale computing clusters, which are allocated based on job parameter settings. Resources are isolated by job. Billing of public resource pools is based on the resource specifications, duration, and instance quantity, regardless of the tasks (including training, deployment, and development) where the public resource pools are used. Public resource pools are provided by ModelArts by default and do not need to be created or configured. You can directly select a public resource pool during AI development.

- **Dedicated Resource Pool**: provides exclusive compute resources, which can be used for notebook instances, training jobs, and model deployment. It delivers higher efficiency and cannot be shared with other users. Buy a dedicated resource pool and select the dedicated resource pool during AI development. For details about the dedicated resource pool, see the following:
  - Dedicated Resource Pool
  - Creating a Dedicated Resource Pool
  - Scaling a Dedicated Resource Pool
  - Deleting a Dedicated Resource Pool

Dedicated Resource Pool

- Dedicated resource pools can be used in the following jobs and tasks: notebook instances, training, TensorBoard, and deployment.
- Dedicated resource pools are classified into two types: Dedicated for Development/Training and Dedicated for Service Deployment. The Dedicated for Development/Training type can be used only for notebook instances, training, and TensorBoard. The Dedicated for Service Deployment type can be used only for model deployment.
- Dedicated resource pools are available only when they are in the Running status. If a dedicated resource pool is unavailable or abnormal, rectify the fault before using it.
After a dedicated resource pool is created, the billing starts based on the selected specifications.

Dedicated resource pools can be billed in pay-per-use or yearly/monthly mode.

Creating a Dedicated Resource Pool

1. Log in to the ModelArts management console and choose Dedicated Resource Pools on the left.
3. Click Create in the upper left corner. The page for creating a dedicated resource pool is displayed.
4. Set the parameters on the page. For details about how to set parameters, see Table 8-1 and Table 8-2.

Table 8-1 Parameters of the Dedicated for Development/Training type

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Type</td>
<td>The default value is Dedicated for Development/Training and cannot be changed.</td>
</tr>
<tr>
<td>Billing Mode</td>
<td>Select a billing mode, Yearly/Monthly or Pay-per-use.</td>
</tr>
<tr>
<td>Name</td>
<td>Name of a dedicated resource pool.</td>
</tr>
<tr>
<td></td>
<td>The value can contain uppercase and lowercase letters, digits, hyphens (-), and underscores (_).</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description of a dedicated resource pool.</td>
</tr>
<tr>
<td>Nodes</td>
<td>Select the number of nodes in a dedicated resource pool. More nodes mean higher computing performance and a higher cost.</td>
</tr>
<tr>
<td>Specifications</td>
<td>Required specifications. The GPU delivers better performance, and the CPU is more cost-effective. If a flavor is sold out, you can purchase it again only after other users delete the resource pool.</td>
</tr>
<tr>
<td>Required Duration</td>
<td>Select the time length when you want to use the resource pool. This parameter is mandatory only when the Yearly/Monthly billing mode is selected.</td>
</tr>
<tr>
<td></td>
<td>The duration ranges from one month to one year. ModelArts provides a 1-year preference package, which allows you to enjoy the product for 1 year by paying for only 10 months.</td>
</tr>
<tr>
<td></td>
<td>• Auto renewal. With auto-renewal, the system automatically renews your product before the product expires.</td>
</tr>
</tbody>
</table>
Table 8-2 Parameters of the **Dedicated for Service Deployment** type

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Type</td>
<td>The default value is <strong>Dedicated for Service Deployment</strong> and cannot be changed.</td>
</tr>
<tr>
<td>Billing Mode</td>
<td>Select a billing mode, <strong>Yearly/Monthly</strong> or <strong>Pay-per-use</strong>. The <strong>Yearly/Monthly</strong> billing mode is supported only in CN North-Beijing4.</td>
</tr>
<tr>
<td>Name</td>
<td>Name of a dedicated resource pool. The value can contain uppercase and lowercase letters, digits, hyphens (-), and underscores (_).</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description of a dedicated resource pool.</td>
</tr>
<tr>
<td>Custom Network Configuration</td>
<td>If you enable <strong>Custom Network Configuration</strong>, the service instance runs on the specified network and can communicate with other cloud service resource instances on the network. If you do not enable <strong>Custom Network Configuration</strong>, ModelArts allocates a dedicated network to each user and isolates users from each other. If you enable <strong>Custom Network Configuration</strong>, set <strong>VPC</strong>, <strong>Subnet</strong>, and <strong>Security Group</strong>. If no network is available, go to the VPC service and create a network.</td>
</tr>
<tr>
<td>AZ</td>
<td>You can select <strong>Random</strong>, <strong>AZ 1</strong>, <strong>AZ 2</strong>, or <strong>AZ 3</strong> based on site requirements. An AZ is a physical region where resources use independent power supplies and networks. AZs are physically isolated but interconnected through an internal network. To enhance workload availability, create nodes in different AZs.</td>
</tr>
<tr>
<td>Nodes</td>
<td>Select the number of nodes in a dedicated resource pool. More nodes mean higher computing performance and a higher cost.</td>
</tr>
<tr>
<td>Specifications</td>
<td>Required specifications. The GPU delivers better performance, and the CPU is more cost-effective.</td>
</tr>
</tbody>
</table>
| Required Duration          | - Select the time length when you want to use the resource pool. This parameter is mandatory only when the **Yearly/Monthly** billing mode is selected. The duration ranges from one month to 11 months.  
- Auto renewal. With auto-renewal, the system automatically renews your product before the product expires. |

5. After confirming that the specifications are correct, create an dedicated resource pool as prompted. After a dedicated resource pool is created, its status changes to **Running**.
Scaling a Dedicated Resource Pool

After a dedicated resource pool is used for a period of time, you can scale out or in the capacity of the resource pool by increasing or decreasing the number of nodes.

A dedicated resource pool in Yearly/Monthly billing mode does not support scaling. If you buy a dedicated resource pool in Pay-per-use billing mode, you need to pay for the nodes after scaling.

The procedure for scaling is as follows:

1. Go to the dedicated resource pool management page, locate the row that contains the desired dedicated resource pool, and click Scale in the Operation column.
2. On the scaling page, increase or decrease the number of nodes. Increasing the node quantity scales out the resource pool whereas decreasing the node quantity scales in the resource pool. Scale the capacity based on service requirements.
   - During scale-out, select the quota of the current account and increase the number of nodes. Otherwise, the scale-out will fail.
   - During scale-in, switch off the desired node in the Operation column to delete the node. To reduce one node, you need to switch off the node in Node List to delete the node. See Figure 8-1.
3. Click Submit. After the request is submitted, the dedicated resource pool management page is displayed.

Deleting a Dedicated Resource Pool

If a dedicated resource pool is no longer needed during AI service development, you can delete the resource pool to release resources and reduce costs.

- **NOTE**

  - After a dedicated resource pool is deleted, the training jobs, notebook instances, and deployment that depend on the resource pool are unavailable. A dedicated resource pool cannot be restored after being deleted. Exercise caution when deleting a dedicated resource pool.
  - A dedicated resource pool in Yearly/Monthly billing mode cannot be deleted.

1. Go to the dedicated resource pool management page, locate the row that contains the desired dedicated resource pool, and click Delete in the Operation column.
2. In the dialog box that is displayed, click OK.
9 Custom Images

9.1 Custom Image Overview

ModelArts provides multiple frequently-used built-in engines. However, when users have special requirements for the deep learning engine and development library, the built-in AI engines cannot meet user requirements. ModelArts provides the custom image function to allow users to customize engines.

The bottom layer of ModelArts uses the container technology. Custom images refer to that users create container images and run them on ModelArts. The custom image function supports command line parameters and environment variables in free-text format. The custom images are highly flexible and support the job boot requirements of any computing engine.

The following services are also required for creating a custom image: Software Repository for Container (SWR), OBS, and Elastic Cloud Server (ECS).

Application Scenarios of Custom Images

- **For Training Models**
  If you have developed a model or training script locally and the AI engine you use is not supported by ModelArts, you can create a custom image based on the basic image packages provided by ModelArts and upload the custom image to SWR. Then, you can use the custom image to create a training job on ModelArts and use the resources provided by ModelArts to train models.

- **For Importing Models**
  If you use an AI engine that is not supported by ModelArts to develop a model, you can create a custom image, import the image to ModelArts for unified management, and deploy the model as a service.

9.2 Creating and Uploading a Custom Image

ModelArts allows you to use custom images to create training jobs and import models. Before creating and uploading a custom image, understand the following information:
• Software Repository for Container (SWR)

SWR provides easy, secure, and reliable management over Docker container images throughout their lifecycle, facilitating the deployment of containerized applications. You can push, pull, and manage container images through SWR console, SWR APIs, or community Command Line Interface (CLI).

Obtain the custom images used by ModelArts for model training and import from the SWR service management list. Upload the custom images you create to SWR.

• Specifications for custom images. For details about how to use a custom image for a training job, see Specifications for Custom Images Used for Training Jobs. For details about how to use a custom image for model import, see Specifications for Custom Images Used for Importing Models.

Creating and Upload a Custom Image

1. Purchase a HUAWEI CLOUD ECS or local host to set up the Docker environment.
2. Obtain the basic image from the local environment.
3. Compile a Dockerfile based on your requirements to build a custom image. For details about how to efficiently compile a Dockerfile, see Software Repository for Container Best Practices.
   - For details about how to create a custom image for a training job, see Example: Creating a Training Job Using a Custom Image.
   - For details about how to use a custom image for a training job, see Building an Image Locally.
4. After creating a custom image, upload it to SWR. For details, see Uploading an Image Through a Container Engine Client.

9.3 For Training Models

9.3.1 Specifications for Custom Images Used for Training Jobs

When creating an image using locally developed models and training scripts, ensure that they meet the specifications defined by ModelArts.

Specifications

• Custom images cannot contain malicious code.
• Part of content in the basic images cannot be changed, including all the files in `/bin`, `/sbin`, `/usr`, and `/lib(64)`, some important configuration files in `/etc`, and the ModelArts tools in `$HOME`.
• A file cannot be added whose owner is root and has permission setuid or setgid.
• The size of a custom image cannot exceed 9.5 GB.
• To ensure that the log content can be displayed normally, the logs must be standard output.
Basic Image Package

To facilitate code download, training log output, and log file upload to OBS, ModelArts provides basic image packages for creating custom images. The basic images provided by ModelArts have the following features:

- Some necessary tools are available in the basic image. You need to create a custom image based on the basic images provided by ModelArts.
- ModelArts continuously updates the basic image versions. For compatible updates, after the basic images are updated, you can still use the old images. For incompatible updates, the custom images created based on the old version cannot run on ModelArts, but the approved custom images can still be used.
- If a custom image fails to be approved and the audit log contains an error message indicating that the basic image does not match, you need to use a new basic image to create an image.
- Table 9-1 and Table 9-2 list components and tools contained in basic images. For details about the complete basic image content, see Dockerfile.

### Table 9-1 Component list

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_train.sh</td>
<td>Training boot script. You can download the code directory, run training commands, redirect training log output, and upload log files to OBS after training commands are executed.</td>
</tr>
</tbody>
</table>

### Table 9-2 Tool list

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>utils.sh</td>
<td>Tool script. The run_train.sh script depends on this script. It provides methods such as SK decryption, code directory download, and log file upload.</td>
</tr>
<tr>
<td>ip_mapper.py</td>
<td>Script for obtaining NIC addresses. By default, the IP address of the ib0 NIC is obtained. Training code can use the IP address of the ib0 NIC to accelerate network communications.</td>
</tr>
<tr>
<td>dls-downloader.py</td>
<td>OBS download script. The utils.sh script depends on this script.</td>
</tr>
</tbody>
</table>

The name format of the basic images provided by ModelArts is as follows:

- CUDA 8/9/92 image
  swr.<region>.myhuaweicloud.com/<image org>/custom-<processor type>-<cuda version>-base:<image tag>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Possible Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;region&gt;</td>
<td>● ap-southeast-1</td>
<td>Region where the image resides. The possible values are described as follows: ● AP-Hong Kong</td>
</tr>
<tr>
<td>&lt;image org&gt;</td>
<td>modelarts-job-dev-image</td>
<td>Organization to which the image belongs. Use modelarts-job-dev-image.</td>
</tr>
<tr>
<td>&lt;processor type&gt;</td>
<td>● cpu ● gpu</td>
<td>Processor type.</td>
</tr>
<tr>
<td>&lt;cuda version&gt;</td>
<td>● cuda92 ● cuda9 ● cuda8</td>
<td>CUDA version installed in the image. In versions earlier than CUDA 10, the CUDA version takes effect only when &lt;processor type&gt; is set to gpu.</td>
</tr>
<tr>
<td>&lt;image tag&gt;</td>
<td>● 1.0 ● 1.1 ● 1.2 ● 1.3</td>
<td>Image version.</td>
</tr>
</tbody>
</table>

- Image of the CUDA 8, 9, or 92 version. MoXing is pre-installed by default. swr.<region>.myhuaweicloud.com/<image org>/custom-<processor type>-<cuda version>-inner-moxing-<python version>:<image tag>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Possible Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;cuda version&gt;</td>
<td>• cuda92</td>
<td>CUDA version installed in the image.</td>
</tr>
<tr>
<td></td>
<td>• cuda9</td>
<td>In versions earlier than CUDA 10, the CUDA version takes effect only when &lt;processor type&gt; is set to gpu.</td>
</tr>
<tr>
<td></td>
<td>• cuda8</td>
<td>NOTE: Check the CUDA version. After the version is specified, it cannot be changed. Otherwise, the training will fail.</td>
</tr>
<tr>
<td>python version</td>
<td>• cp27</td>
<td>Python environment.</td>
</tr>
<tr>
<td></td>
<td>• cp36</td>
<td></td>
</tr>
<tr>
<td>&lt;image tag&gt;</td>
<td>1.3</td>
<td>Image version.</td>
</tr>
</tbody>
</table>

- The image of CUDA 10.0, 10.1, or 10.2 uses Ubuntu 18.04 as the basic image. MoXing is pre-installed by default.

```
swr.<region>.myhuaweicloud.com/<image org>/custom-base/<cuda version>-<python version>-<os>-<arch>:<image tag>
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Possible Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;region&gt;</td>
<td>• ap-southeast-1</td>
<td>Region where the image resides. The possible values are described as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• AP-Hong Kong</td>
</tr>
<tr>
<td>&lt;image org&gt;</td>
<td>modelarts-job-dev-image</td>
<td>Organization to which the image belongs. Use modelarts-job-dev-image.</td>
</tr>
<tr>
<td>&lt;cuda version&gt;</td>
<td>• cuda10.0</td>
<td>CUDA version installed in the image.</td>
</tr>
<tr>
<td></td>
<td>• cuda10.1</td>
<td>NOTE: Check the CUDA version. After the version is specified, it cannot be changed. Otherwise, the training will fail.</td>
</tr>
<tr>
<td></td>
<td>• cuda10.2</td>
<td></td>
</tr>
<tr>
<td>python version</td>
<td>cp36</td>
<td>Python 3.6 environment.</td>
</tr>
<tr>
<td>os</td>
<td>ubuntu18.04</td>
<td>Operating system.</td>
</tr>
<tr>
<td>arch</td>
<td>x86</td>
<td>Architecture.</td>
</tr>
<tr>
<td>&lt;image tag&gt;</td>
<td>1.1</td>
<td>Image version.</td>
</tr>
</tbody>
</table>
9.3.2 Creating a Training Job Using a Custom Image

After creating and uploading a custom image to SWR, you can use the image to create a training job on the ModelArts management console to complete model training.

**Prerequisites**

- You have created a custom image package based on ModelArts specifications. For details about the specifications you need to comply with when using a custom image to create training jobs, see Specifications for Custom Images Used for Training Jobs.
- You have uploaded the custom image to SWR. For details, see Creating and Uploading a Custom Image.

**Creating a Training Job**

Log in to the ModelArts management console and create a training job according to Creating a Training Job. When using a custom image to create a job, pay attention to the settings of Algorithm Source, Environment Variable, and Resource Pool.

- **Algorithm Source**
  
  Select **Custom**.

  - **Image Path**: SWR URL after the image is uploaded to SWR

  ![SWR image address](image)

- **Code Directory**: OBS path for storing the training code file.

  - **Boot Command**: boot command after the image is started. The basic format is as follows:

    ```bash
    /home/work/run_train.sh {UserCommand}
    /home/work/run_train.sh [python/bash/..] {file_location} {file_parameter}
    run_train.sh is the training boot script. After this script is executed, ModelArts recursively downloads all content in the code directory to the local path of the container. The local path is in the format of /home/work/user-job-dir/$\{Name of the last level in the code directory\}.
    
    For example, if the OBS path of the training code file is obs://obs-bucket/new/train.py and the code directory is obs://obs-bucket/new/, the local path of the container is /home/work/user-job-dir/new/. The
local training code path of the container is /home/work/user-job-dir/new/train.py. Then, you can set the boot command to the following:

```
bash /home/work/run_train.sh python /home/work/user-job-dir/new/train.py {python_file_parameter}
```

**NOTE**

If you create a training job using a custom image, ModelArts allows you to customize the boot command. The following are two basic formats for the boot command:

```
bash /home/work/run_train.sh {UserCommand}
bash /home/work/run_train.sh [python/bash/..] {file_location} {file_parameter}
```

run_train.sh is the training boot script. When creating a custom image, you can implement the training boot script or place the training code in the custom image environment in advance to customize the boot command (in the basic formats or any other formats).

- **Environment Variable**

  After the container is started, besides the environment variables added by configuring Environment Variable during training job creation, Table 9-3 lists other environment variables to be loaded. You can determine whether to use these environment variables in your own Python training script, or run the {python_file_parameter} command to pass the required parameters.

**Table 9-3** Optional environment variables

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLS_TASK_INDEX</td>
<td>Container index, starting from 0.</td>
</tr>
<tr>
<td>DLS_TASK_NUMBER</td>
<td>Number of containers, corresponding to Compute Nodes</td>
</tr>
<tr>
<td>DLS_APP_URL</td>
<td>Code directory, corresponding to Code Dir with the protocol name added. For example, you can use $DLS_APP_URL/*.py to read files in OBS.</td>
</tr>
<tr>
<td>DLS_DATA_URL</td>
<td>Dataset path, corresponding to Data Source with the protocol name added</td>
</tr>
<tr>
<td>DLS_TRAIN_URL</td>
<td>Training output path, corresponding to Training Output Path with the protocol name added</td>
</tr>
</tbody>
</table>
### Environment Variable

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATCH_{jobName}.0_HOSTS (standalone)</td>
<td>For standalone training, that is, when the number of compute nodes is 1, the environment variable is BATCH_{jobName}.0_HOSTS. The format of the HOSTS environment variable is hostname:port. A container can view the HOSTS of all containers in the same job, such as BATCH_CUSTOM0_HOSTS and BATCH_CUSTOM1_HOSTS, varying according to the indexes. If the resource pool is a dedicated resource pool with the 8GPU specifications, the network type of the container is a host network, and the host IB network can be used to accelerate communications. If other resource pools are used, the network is a container network. <strong>NOTE</strong> When the host IB network is used for communication acceleration, the ip_mapper.py tool is required to obtain the IP address of the ib0 NIC for using the IPoIB feature.</td>
</tr>
</tbody>
</table>

- **Resource Pool**
  
  If you select a resource pool of the GPU type, ModelArts mounts NVME SSDs to the /cache directory. You can use this directory to store temporary files.

![Figure 9-2 Creating a training job](image)

**Running a Training Job Created Using a Custom Image**

After a custom image is uploaded to SWR, ModelArts is authorized to obtain and run the image by default when you create a training job using the custom image. When a custom image is run for the first time, the image is checked first. For details about the check content, see Specifications for Custom Images Used for Training Jobs. The check failure cause is outputted in the log, and you can modify the image based on the log.
After the image is checked, the backend starts the custom image container to run the training job. You can view the training status based on the log.

**NOTE**

After an image is reviewed, the image does not need to be reviewed again when being used to create training jobs again.

### 9.3.3 Example: Creating a Training Job Using a Custom Image

The files required in this example are stored in GitHub. In this example, the **Mnist dataset** is used.

- **mnist_softmax.py**: standalone training script

#### Creating and Uploading a Custom Image

In this example, the Dockerfile file is used to customize an image.

A Linux x86_x64 host is used here. You can purchase an ECS of the same specifications or use an existing local host to create a custom image.

1. **Install Docker.** For details, see [https://docs.docker.com/engine/install/binaries/#install-static-binaries](https://docs.docker.com/engine/install/binaries/#install-static-binaries).
   
   The following uses the Linux x86_64 architecture as an example to describe how to obtain the **Docker 19.03.2** installation package. Run the following command to install the Docker software:

   ```bash
tar -xzf docker-19.03.2.tgz
   cp docker/* /usr/bin/
   ```

   If the **docker images** command is successfully executed, Docker has been installed. In this case, skip this step.

2. **Obtain a basic image.**
A custom image used for a training job must be compiled based on a basic image. For details about the formats of basic image names, see Basic Image Package. Run the following command to obtain a basic image for custom images:

```
docker pull swr.<region>.myhuaweicloud.com/<image org>/custom-<processor type>-[<cuda version>]-base:<image tag>
```

In addition, you can run the `docker images` command to view the local image list.

3. (Optional) Modify the basic image for debugging. If you need to modify the PyPi mirror and apt source based on the network status, perform the following steps to modify the file content in the container:

The following uses the basic image `swr.cn-north-4.myhuaweicloud.com/modelarts-job-dev-image/custom-cpu-base:1.3` as an example.

a. Run the following command to start the container and allocate a pseudo TTY to connect to the stdin of the container:

```
docker run -ti ${Container image} bash
```

b. The context is in the container.

i. Configure the PyPi mirror in the basic image based on the network status to accelerate the download of Python packages. The PyPi mirror is set to the one provided by Huawei DevCloud by default. You can run the following command in the container to view the PyPi mirror. If you want to use another PyPi mirror, change the value of `index-url` in the command to the value of your desired PyPi mirror.

```
cat /root/.pip/pip.conf
```

```
[global]
index-url = http://repo.myhuaweicloud.com/repository/pypi/simple
format = columns

[install]
trusted-host=repo.myhuaweicloud.com
```

ii. Configure the Ubuntu16.04 apt source in the basic image based on the network status to accelerate the download of apt packages. The apt source in the basic image is provided by Ubuntu 16.04. Run the following command in the container to change the default source to the apt source of DevCloud:

```
If you want to use another apt source, change http://repo.myhuaweicloud.com in the command to your desired source.
```

```
cp -a /etc/apt/sources.list /etc/apt/sources.list.bak
sed -i "s@http://.*archive.ubuntu.com@http://repo.myhuaweicloud.com@g" /etc/apt/sources.list
sed -i "s@http://.*security.ubuntu.com@http://repo.myhuaweicloud.com@g" /etc/apt/sources.list
```

c. After the modification, run the following command to exit the pseudo TTY:

```
exit
```

d. The context is in the host. Run the following command to save the modified container as an image:

```
docker commit ${container-id} swr.cn-north-4.myhuaweicloud.com/modelarts-job-dev-image/custom-cpu-base:1.3-updated
```

In the preceding command, ${container-id} indicates the ID of the modified container, which can be obtained by running the `docker ps -a` command. The modified container is named `swr.cn-north-4.myhuaweicloud.com/modelarts-job-dev-image/custom-cpu-`
If you need to use the image of the modified container for build tasks, use the new container image name.

4. Compile a Dockerfile for building a custom Python 3.6.5 + TensorFlow 1.13.2 image.

The file name is **tf-1.13.2.dockerfile**. In this Dockerfile example, Minicoda and Python are reinstalled. Therefore, you need to reinstall the Python dependency package. The basic image contains the following default Python packages:

- boto3 (1.7.29)
- botocore (1.10.81)
- docutils (0.14)
- futures (3.2.0)
- jmespath (0.9.3)
- netifaces (0.10.7)
- python-dateutil (2.7.5)
- pyzmq (17.1.2)
- s3transfer (0.1.13)
- setuptools (20.7.0)
- six (1.11.0)
- wheel (0.29.0)

In this example, Boto3 is installed. You can add the installation of other dependency packages based on site requirements. For details about how to compile the Dockerfile file, see the **Dockerfile Reference**.

```
FROM swr.cn-north-4.myhuaweicloud.com/modelarts-job-dev-image/custom-cpu-base:1.3
RUN mkdir -p /tmp/install && cd /tmp/install && \
    curl -o Miniconda3-4.5.4-Linux-x86_64.sh -k https://repo.anaconda.com/miniconda/Miniconda3-4.5.4-\n    Linux-x86_64.sh && \
    root/miniconda3/bin/pip install \
    tf-1.13.2 && \
    pip install boto3==1.7.29 && \
    rm -rf /tmp/install \
ENV PATH=/root/miniconda3/bin:/$PATH
```

5. Create a custom image.

In the following example, the image is in the **cn-north-4** region and belongs to the **deep-learning-diy** organization. Run the following command in the directory where the **tf-1.13.2.dockerfile** file resides:

```
docker build -f tf-1.13.2.dockerfile . -t swr.cn-north-4.myhuaweicloud.com/deep-learning-diy/\n    tf-1.13.2:latest
```

6. Push the image to SWR. For details about how to upload an image, see the **Software Repository for Container User Guide**.

The prerequisite is that you have created an organization and obtained the **SWR login command**. In the following example, the image is located in the **cn-north-4** region and belongs to the **deep-learning-diy** organization. Run the following command to push the image to SWR:

```
docker push swr.cn-north-4.myhuaweicloud.com/deep-learning-diy/tf-1.13.2:latest
swr.cn-north-4.myhuaweicloud.com/deep-learning-diy/tf-1.13.2:latest is the SWR URL of the custom image.
```
Standalone Training

1. Upload training code `mnist_softmax.py` and training data to OBS. Store the code and data in the code root directory so that they can be directly downloaded to the container.

   The root directory `obs://deep-learning/new/mnist/` is used as an example.
   The training code file is `obs://deep-learning/new/mnist/.
   The data is stored in `obs://deep-learning/new/mnist/minist_data`.

2. Create a training job using a custom image. Set Data Storage Location and Training Output Path based on site requirements. Set Image Path, Code Directory, and Boot Command as follows:
   - **Image Path**: Enter the SWR URL of the uploaded image.
   - **Code Directory**: Enter the OBS path for storing the training code, that is, the code root directory in 1.

   Before a training job is started, ModelArts automatically recursively downloads all content in the code directory to the local path of the container. The local path of the container is `/home/work/user-job-dir/$ {Last level of the code root directory}/`. For example, if Code Directory is set to `obs://deep-learning/new/mnist`, the local path is `/home/work/user-job-dir/mnist/`, and the code boot file is `/home/work/user-job-dir/mnist/mnist_softmax.py`.

   - **Boot Command**: `bash /home/work/run_train.sh python /home/work/user-job-dir/mnist/mnist_softmax.py --data_url /home/work/user-job-dir/mnist/mnist_data`

   `/home/work/user-job-dir/mnist/mnist_softmax.py` is the code boot file, and `--data_url /home/work/user-job-dir/mnist/mnist_data` is the data storage path.

3. After the training job is created, the code directory is downloaded, the custom image is reviewed, and the training job is completed in the background. Generally, training jobs run for a period of time, which may be several minutes or tens of minutes depending on the amount of data and resources you select. After the program is executed successfully, the log similar to the following is outputted:

   ![Figure 9-5 Run log information](image)

9.4 For Importing Models
9.4.1 Specifications for Custom Images Used for Importing Models

When creating an image using locally developed models, ensure that they meet the specifications defined by ModelArts.

Specifications for Custom Images Used for Model Management

- Custom images cannot contain malicious code.
- The size of a custom image cannot exceed 10 GB.
- **External port of images**
  The external service port of the image must be 8080. The inference interface must be consistent with the URL defined by `apis` in the `config.json` file. The inference interface can be directly accessed when the image is started. The following is an example of accessing the mnist image. The image contains the model trained with the mnist dataset. The model can identify handwritten digits in images. In this example, `listen_ip` indicates the IP address of the container.
  - Sample request: `curl -X POST \ http://{listen_ip}:8080/ \ -F images=@seven.jpg`
  - Sample response
    `{"mnist_result": 7}`
- **Health check port**
  A custom image must provide a health check interface for ModelArts to call. The health check interface is configured in the `config.json` file. For details, see the model configuration file compilation description. A sample health check interface is as follows:
  - **URI**
    `GET /health`
  - Sample request: `curl -X GET \ http://{listen_ip}:8080/health`
  - Sample response
    `{"health": "true"}`
  - Status code

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
<td>Successful request</td>
</tr>
</tbody>
</table>

- **Log file output**
  To ensure that the log content can be displayed normally, the logs must be standard output.
- **Image boot file**
  To deploy a batch service, set the boot file of an image to `/home/run.sh` and use CMD to set the default boot path. The following is a sample Dockerfile.

```
CMD /bin/sh /home/run.sh
```
● **Image dependencies**
  To deploy a batch service, install component packages such as Python, JRE/JDK, and ZIP in the image.

### 9.4.2 Importing a Model Using a Custom Image

After creating and uploading a custom image to SWR, you can use the image to import a model and deploy the model as a service on the ModelArts management console.

#### Prerequisites

- You have created a custom image package based on ModelArts specifications. For details about how to create a custom image package, see [Building an Image Locally](#). For details about the specifications you need to comply with when using a custom image to import a model, see [Specifications for Custom Images Used for Importing Models](#).
- You have uploaded the custom image to SWR. For details, see [Verifying the Image on the Local Host and Uploading the Image to SWR](#).

#### Importing a Model

Set basic parameters for importing a model according to [Importing a Meta Model from a Container Image](#). When importing a model using a custom image, pay attention to the settings of **Meta Model Source** and **Configuration File**.

- **Meta Model Source**
  Select **Container Image**. Click ![Image Path](image.png) in the edit box of **Container Image Path** to select an image. The system automatically lists all images uploaded to SWR. Select an image based on the site requirements.

- **Configuration File**
  The model configuration file needs to be compiled independently. For details about how to compile the model configuration file, see [Specifications for Compiling the Model Configuration File](#). For details about the configuration file examples of a custom image, see [Example of the Custom Image Model Configuration File](#). After editing the model configuration file based on the ModelArts specifications, upload it to OBS or use **Edit online** on the **Import Model** page.

![Figure 9-6 Importing a model using a custom image](image.png)
Deploying a Service

After a model is successfully imported using a custom image, that is, the model status is normal, you can deploy the model as a service. On the Models page, click Deploy in the Operation column and select a service type, for example, Real-time Service.

You can deploy models as real-time or batch services based on the business logic of your custom image. The procedure for deploying a model imported using other methods is the same as that for deploying a model imported using a custom image. For details, see Model Deployment Overview.

9.4.3 Example: Importing a Model Using a Custom Image

For AI engines that are not supported by ModelArts, you can import the models you compile to ModelArts from custom images. This section describes how to use a custom image to import a model.

Building an Image Locally

A Linux x86_x64 host is used here. You can purchase an ECS of the same specifications or use an existing local host to create a custom image.

1. Install Docker. For details, see Docker Documentation. You can install Docker as follows:
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh

2. Obtain basic images. Ubuntu18.04 is used in this example.
   docker pull ubuntu:18.04

3. Create the self-define-images folder, and compile the Dockerfile file and the test_app.py application service code for the custom image in the folder. In this sample code, the application service code uses the Flask framework.

The file structure is as follows:

```
self-define-images/
  --Dockerfile
  --test_app.py
```

```
Dockerfile
From ubuntu:18.04
# Change the apt source to the HUAWEI CLOUD source and install Python, Python3-PIP, and Flask.
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y python3 python3-pip && \

# Copy the application service code to the image.
COPY test_app.py /opt/test_app.py

# Specify the boot command of the image.
CMD python3 /opt/test_app.py
```

```
test_app.py
from flask import Flask, request
import json
app = Flask(__name__)
```
@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("----------- in hello func ----------")
data = json.loads(request.get_data(as_text=True))
print(data)
username = data['name']
rsp_msg = "Hello, {}!".format(username)
return json.dumps({'response':rsp_msg, 'indent':4})

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("----------- in goodbye func ----------")
    return 'Goodbye!'

@app.route('/', methods=['POST'])
def default_func():
    print("----------- in default func ----------")
data = json.loads(request.get_data(as_text=True))
return " called default func !
{} 
".format(str(data))

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080)

## NOTE

ModelArts forwards requests to port 8080 of the service started from the custom image. Therefore, the service listening port in the container must be port 8080. See the test_app.py file.

4. Go to the self-define-images folder and run the following command to create custom image test:v1:
docker build -t test:v1 .

5. You can run docker image to view the custom image you have created.

Verifying the Image on the Local Host and Uploading the Image to SWR

1. Run the following command in the local environment to start the custom image:
docker run -it -p 8080:8080 test:v1

2. Open another terminal and run the following commands to verify the functions of the three APIs of the custom image:
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/goodbye
curl -X GET 127.0.0.1:8080/goodbye

If information similar to the following is displayed, the the function verification is successful.
3. Upload the custom image to SWR. For details, see Software Repository for Container User Guide.

4. After the custom image is uploaded, you can view the uploaded image on the My Images > Private Images page of the SWR console.

Importing a Model from the Container Image

For details, see Importing a Meta Model from a Container Image. Note the following parameters:

- **Meta Model Source**: Select Container image.
- **Container Image Path**: Select the created private image.

- **Configuration File**: Select Edit Online. For details about the configuration file requirements, see Specifications for Compiling the Model Configuration File. Click Save.

The configuration file is as follows:

```json
{
  "model_algorithm": "test_001",
  "model_type": "Image",
  "apis": [{
    "protocol": "http",
    "url": "/",
    "method": "post",
    "request": {
      "Content-type": "application/json"
    },
    "response": {
      "Content-type": "application/json"
    }
  },
  { "protocol": "http",
```
Deploying the Model as a Real-Time Service

1. Deploy the model as a real-time service. For details, see Deploying a Model as a Real-Time Service.
2. View the details about the real-time service.

Figure 9-11 Usage Guides

3. Access the real-time service on the Predictions tab page.
Figure 9-12 Accessing a real-time service
10 Model Package Specifications

10.1 Model Package Specifications

When you import models in Model Management, if the meta model is imported from OBS or a container image, the model package must meet the following specifications:

- The model package must contain the model directory. The model directory stores the model file, model configuration file, and model inference code.
- The model configuration file must exist and its name is fixed to config.json. There exists only one model configuration file. For details about how to compile the model configuration file, see Specifications for Compiling the Model Configuration File.
- The model inference code file is optional. If this file is required, the file name is fixed to customize_service.py. There must be one and only one such file. For details about how to compile the model inference code, see Specifications for Compiling Model Inference Code. The .py file on which customize_service.py depends can be directly stored in the model directory. You are advised to use the Python import mode to import the custom package. The other files on which customize_service.py depends can be stored in the model directory. You must use absolute paths to access these files.

ModelArts provides samples and sample code for multiple engines. You can compile your configuration files and inference code by referring to ModelArts Samples. ModelArts also provides custom script examples of common AI engines. For details, see Examples of Custom Scripts.

Model Package Example

- Structure of the TensorFlow-based model package

  When publishing the model, you only need to specify the ocr directory.

  OBS bucket/directory name

  ├── ocr
  │   └── model (Mandatory) Name of a fixed subdirectory, which is used to store model-related files
  │          └── <<Custom Python package>> (Optional) User's Python package, which can be directly referenced in the model inference code
  │          └── saved_model.pb (Mandatory) Protocol buffer file, which contains the diagram description
of the model
|   |   ├── variables Name of a fixed sub-directory, which contains the weight and deviation rate of the model. It is mandatory for the main file of the *.pb model.
|   |   │   ├── variables.index Mandatory
|   |   │   ├── variables.data-00000-of-00001 Mandatory
|   |   │   └── config.json (Mandatory) Model configuration file. The file name is fixed to config.json.

Only one model configuration file exists.
|   |   └── customize_service.py (Optional) Model inference code. The file name is fixed to customize_service.py. Only one model inference code file exists. The files on which customize_service.py depends can be directly stored in the model directory.

- Structure of the MXNet-based model package

When publishing the model, you only need to specify the resnet directory.

OBS bucket/directory name
|-- resnet
   |-- model (Mandatory) Name of a fixed subdirectory, which is used to store model-related files
   |   ├── <<Custom Python package>> (Optional) User's Python package, which can be directly referenced in the model inference code
   |   ├── resnet-50-symbol.json (Mandatory) Model definition file, which contains the neural network description of the model.
   |   ├── resnet-50-0000.params (Mandatory) Model variable parameter file, which contains parameter and weight information
   |   ├── config.json (Mandatory) Model configuration file. The file name is fixed to config.json.

Only one model configuration file exists.
|   |   └── customize_service.py (Optional) Model inference code. The file name is fixed to customize_service.py. Only one model inference code file exists. The files on which customize_service.py depends can be directly stored in the model directory.

- Structure of the Image-based model package

When publishing the model, you only need to specify the resnet directory.

OBS bucket/directory name
|-- resnet
   |-- model (Mandatory) Name of a fixed subdirectory, which is used to store model-related files
   |   └── config.json (Mandatory) Model configuration file (the address of the SWR image must be configured). The file name is fixed to config.json. Only one model configuration file exists.

- Structure of the PySpark-based model package

When publishing the model, you only need to specify the resnet directory.

OBS bucket/directory name
|-- resnet
   |-- model (Mandatory) Name of a fixed subdirectory, which is used to store model-related files
   |   ├── <<Custom Python package>> (Optional) User's Python package, which can be directly referenced in the model inference code
   |   |   └── spark_model (Mandatory) Model directory, which contains the model content saved by PySpark
   |   |   ├── config.json (Mandatory) Model configuration file. The file name is fixed to config.json.
   |   |   └── customize_service.py (Optional) Model inference code. The file name is fixed to customize_service.py. Only one model inference code file exists. The files on which customize_service.py depends can be directly stored in the model directory.

- Structure of the PyTorch-based model package

When publishing the model, you only need to specify the resnet directory.

OBS bucket/directory name
|-- resnet
   |-- model (Mandatory) Name of a fixed subdirectory, which is used to store model-related files
   |   ├── <<Custom Python package>> (Optional) User's Python package, which can be directly referenced in the model inference code
   |   |   └── resnet50.pth (Mandatory) PyTorch model file, which contains variable and weight information and is saved as state_dict
   |   |   └── config.json (Mandatory) Model configuration file. The file name is fixed to config.json.
   |   └── customize_service.py (Optional) Model inference code. The file name is fixed to customize_service.py. Only one model inference code file exists. The files on which customize_service.py depends can be directly stored in the model directory.
Structure of the Caffe-based model package

When publishing the model, you only need to specify the resnet directory.

OBS bucket/directory name

- resnet
- model (Mandatory) Name of a fixed subdirectory, which is used to store model-related files
- <<Custom Python package>> (Optional) User’s Python package, which can be directly referenced in the model inference code
- deploy.prototxt (Mandatory) Caffe model file, which contains information such as the model network structure
- resnet.caffemodel (Mandatory) Caffe model file, which contains variable and weight information
- config.json (Mandatory) Model configuration file. The file name is fixed to config.json. Only one model configuration file exists.
- customize_service.py (Optional) Model inference code. The file name is fixed to customize_service.py. Only one model inference code file exists. The files on which customize_service.py depends can be directly stored in the model directory.

Structure of the XGBoost-based model package

When publishing the model, you only need to specify the resnet directory.

OBS bucket/directory name

- resnet
- model (Mandatory) Name of a fixed subdirectory, which is used to store model-related files
- <<Custom Python package>> (Optional) User’s Python package, which can be directly referenced in the model inference code
- *.m (Mandatory): Model file whose extension name is .m
- config.json (Mandatory) Model configuration file. The file name is fixed to config.json. Only one model configuration file exists.
- customize_service.py (Optional) Model inference code. The file name is fixed to customize_service.py. Only one model inference code file exists. The files on which customize_service.py depends can be directly stored in the model directory.

Structure of the Scikit_Learn-based model package

When publishing the model, you only need to specify the resnet directory.

OBS bucket/directory name

- resnet
- model (Mandatory) Name of a fixed subdirectory, which is used to store model-related files
- <<Custom Python package>> (Optional) User’s Python package, which can be directly referenced in the model inference code
- *.m (Mandatory): Model file whose extension name is .m
- config.json (Mandatory) Model configuration file. The file name is fixed to config.json. Only one model configuration file exists.
- customize_service.py (Optional) Model inference code. The file name is fixed to customize_service.py. Only one model inference code file exists. The files on which customize_service.py depends can be directly stored in the model directory.

10.2 Specifications for Compiling the Model Configuration File

A model developer needs to compile a configuration file when publishing a model. The model configuration file describes the model usage, computing framework, precision, inference code dependency package, and model API.

Configuration File Parameter Description

The configuration file is in JSON format. Table 10-1 describes the parameters.
### Table 10-1 Parameter description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_algorithm</td>
<td>Yes</td>
<td>String</td>
<td>Model algorithm, which is set by the model developer to help model users understand the usage of the model. The value must start with a letter and contain no more than 36 characters. Chinese characters and special characters (,!&quot;&lt;&gt;) are not allowed. Common model algorithms include <strong>image_classification</strong> (image classification), <strong>object_detection</strong> (object detection), and <strong>predict_analysis</strong> (prediction analysis).</td>
</tr>
<tr>
<td>model_type</td>
<td>Yes</td>
<td>String</td>
<td>Model AI engine, which indicates the computing framework used by a model. The options are <strong>TensorFlow</strong>, <strong>MXNet</strong>, <strong>Spark_MLlib</strong>, <strong>Caffe</strong>, <strong>Scikit_Learn</strong>, <strong>XGBoost</strong>, <strong>Image</strong>, and <strong>PyTorch</strong>. <strong>Image</strong> is not a common AI framework. When model_type is set to <strong>Image</strong>, a model is imported from a custom image. In this case, <strong>swr_location</strong> is mandatory. For details about how to make Image images, see <a href="#">Custom Image Specifications</a>.</td>
</tr>
<tr>
<td>runtime</td>
<td>No</td>
<td>String</td>
<td>Model runtime environment. Python 2.7 is used by default. The value of runtime depends on the value of model_type. If model_type is set to <strong>Image</strong>, you do not need to set runtime. If model_type is set to another frequently-used framework, select the engine and development environment. For details about the supported running environments, see Table 6-8. If your model needs to run on a specified CPU or GPU, select the runtime based on the suffix information. If the runtime does not contain the CPU or GPU information, read the description of each runtime in Table 6-8.</td>
</tr>
<tr>
<td>swr_location</td>
<td>No</td>
<td>String</td>
<td>SWR image address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- If you import a custom image model from a container image, you do not need to set swr_location.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- If you import a custom image model from OBS (not recommended) and set model_type to <strong>Image</strong>, you must set swr_location. swr_location indicates the address of the Docker image on SWR, indicating that the Docker image on SWR is used to publish the model.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Mandatory</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>metrics</td>
<td>No</td>
<td>Object</td>
<td>Model precision information, including the average value, recall rate, precision, and accuracy. For details about the metrics object structure, see Table 10-2. This parameter is used only to display model information and is optional.</td>
</tr>
<tr>
<td>apis</td>
<td>Yes</td>
<td>api array</td>
<td>Format of the requests received and returned by a model. The value is structure data. It is the RESTful API array provided by a model. For details about the API data structure, see Table 10-3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• When <code>model_type</code> is set to Image, that is, in the model scenario of a custom image, APIs with different paths can be declared in <code>apis</code> based on the request path exposed by the image.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• When <code>model_type</code> is not Image, only one API whose request path is / can be declared in <code>apis</code> because the preconfigured AI engine exposes only one inference API whose request path is /.</td>
</tr>
<tr>
<td>dependen-</td>
<td>No</td>
<td>dependency array</td>
<td>Package on which the model inference code depends, which is structure data. Model developers need to provide the package name, installation mode, and version constraints. Currently, only the pip installation mode is supported. Table 10-6 describes the dependency array.</td>
</tr>
<tr>
<td>cies</td>
<td></td>
<td></td>
<td>If the model package does not contain the <code>customize_service.py</code> file, you do not need to set this parameter. Dependency packages cannot be installed for custom image models.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>NOTE</strong> The <code>dependencies</code> parameter supports multiple dependency structure arrays in json format and is applicable to scenarios where the installation packages have dependency relationships. For details, see How Do I Describe the Dependency Relationship Between Installation Packages and Model Configuration Files When a Model Is Imported?.</td>
</tr>
<tr>
<td>health</td>
<td>No</td>
<td>health data structure</td>
<td>Configuration of an image health interface. This parameter is mandatory only when <code>model_type</code> is set to Image. For details about the model_type data structure, see Table 10-8.</td>
</tr>
</tbody>
</table>
### Table 10-2 metrics object description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>No</td>
<td>Number</td>
<td>F1 score. The value is rounded to 17 decimal places.</td>
</tr>
<tr>
<td>recall</td>
<td>No</td>
<td>Number</td>
<td>Recall rate. The value is rounded to 17 decimal places.</td>
</tr>
<tr>
<td>precision</td>
<td>No</td>
<td>Number</td>
<td>Precision. The value is rounded to 17 decimal places.</td>
</tr>
<tr>
<td>accuracy</td>
<td>No</td>
<td>Number</td>
<td>Accuracy. The value is rounded to 17 decimal places.</td>
</tr>
</tbody>
</table>

### Table 10-3 api array

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>Yes</td>
<td>String</td>
<td>Request protocol. Set the parameter value to http or https based on your custom image. For details about other parameter, see Example of the Object Detection Model Configuration File.</td>
</tr>
<tr>
<td>url</td>
<td>Yes</td>
<td>String</td>
<td>Request path. For a custom image model (model_type is Image), set this parameter to the actual request path exposed in the image. For a non-custom image model (model_type is not Image), the URL can only be /.</td>
</tr>
<tr>
<td>method</td>
<td>Yes</td>
<td>String</td>
<td>Request method</td>
</tr>
<tr>
<td>request</td>
<td>Yes</td>
<td>Object</td>
<td>Request body. For details about the request structure, see Table 10-4.</td>
</tr>
<tr>
<td>response</td>
<td>Yes</td>
<td>Object</td>
<td>Response body. For details about the response structure, see Table 10-5.</td>
</tr>
</tbody>
</table>
### Table 10-4 request description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Content-type | Yes       | String    | Data is sent based on the specified content type. The options are as follows:  
- **application/json**: sends JSON data.  
- **multipart/form-data**: uploads a file.  
**NOTE** For machine learning models, only **application/json** is supported. |
| data        | Yes       | String    | The request body is described in JSON schema. |

### Table 10-5 response description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Content-type | Yes       | String    | Data is sent based on the specified content type. The options are as follows:  
- **application/json**: sends JSON data.  
- **multipart/form-data**: uploads a file.  
**NOTE** For machine learning models, only **application/json** is supported. |
| data        | Yes       | String    | The response body is described in JSON schema. |

### Table 10-6 dependency array

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>installer</td>
<td>Yes</td>
<td>String</td>
<td>Installation method. Only <strong>pip</strong> is supported.</td>
</tr>
<tr>
<td>packages</td>
<td>Yes</td>
<td>package array</td>
<td>Dependency package collection. For details about the package structure array, see <strong>Table 10-7</strong>.</td>
</tr>
</tbody>
</table>
### Table 10-7 package array

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_name</td>
<td>Yes</td>
<td>String</td>
<td>Dependency package name. Chinese characters and special characters (&amp;!&quot;&lt;&gt;&amp;) are not allowed.</td>
</tr>
<tr>
<td>package_version</td>
<td>No</td>
<td>String</td>
<td>Dependency package version. If the dependency package does not rely on the version number, leave this field blank. Chinese characters and special characters (&amp;!&quot;&lt;&gt;&amp;) are not allowed.</td>
</tr>
<tr>
<td>restraint</td>
<td>No</td>
<td>String</td>
<td>Version restriction. This parameter is mandatory only when package_version is configured. Possible values are EXACT, ATLEAST, and ATMOST.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- <strong>EXACT</strong> indicates that a specified version is installed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- <strong>ATLEAST</strong> indicates that the version of the installation package is not earlier than the specified version.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- <strong>ATMOST</strong> indicates that the version of the installation package is not later than the specified version.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>NOTE</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- If there are specific requirements on the version, preferentially use <strong>EXACT</strong>. If <strong>EXACT</strong> conflicts with the system installation packages, you can select <strong>ATLEAST</strong>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- If there is no specific requirement on the version, you are advised to retain only the <strong>package_name</strong> parameter and leave <strong>restraint</strong> and <strong>package_version</strong> blank.</td>
</tr>
</tbody>
</table>

### Table 10-8 health data structure description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Yes</td>
<td>String</td>
<td>Request URL of the health check interface</td>
</tr>
<tr>
<td>Parameter</td>
<td>Mandatory</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>protocol</td>
<td>No</td>
<td>String</td>
<td>Request protocol of the health check interface. Currently, only HTTP is supported.</td>
</tr>
<tr>
<td>initial_delay_seconds</td>
<td>No</td>
<td>String</td>
<td>After an instance is started, a health check starts after seconds configured in initial_delay_seconds.</td>
</tr>
<tr>
<td>timeout_seconds</td>
<td>No</td>
<td>String</td>
<td>Health check timeout</td>
</tr>
</tbody>
</table>

**Example of the Object Detection Model Configuration File**

The following code uses the TensorFlow engine as an example. You can modify the `model_type` parameter based on the actual engine type.

- **Model input**
  - Key: images
  - Value: image files

- **Model output**

  ```json
  {...
  "detection_classes": [ "face", "arm" ],
  "detection_boxes": [ [ 33.6, 42.6, 104.5, 203.4 ], [ 103.1, 92.8, 765.6, 945.7 ] ],
  "detection_scores": [0.99, 0.73]
  ...}
  ```

- **Configuration file**

  ```json
  {...
  "model_type": "TensorFlow",
  "model_algorithm": "object_detection",
  "metrics": { "f1": 0.345294,
  "accuracy": 0.462963,
  "precision": 0.338977,
  "recall": 0.351852
  },
  "apis": [{
  "protocol": "http",
  "url": "/"},
  ```
Example of the Image Classification Model Configuration File

The following code uses the TensorFlow engine as an example. You can modify the model_type parameter based on the actual engine type.

- Model input
  - Key: images
Value: image files

- Model output

```json
{
  "predicted_label": "flower",
  "scores": [
    ["rose", 0.99],
    ["begonia", 0.01]
  ]
}
```

- Configuration file

```json
{
  "model_type": "TensorFlow",
  "model_algorithm": "image_classification",
  "metrics": {
    "f1": 0.345294,
    "accuracy": 0.462963,
    "precision": 0.338977,
    "recall": 0.351852
  },
  "apis": [{
    "protocol": "http",
    "url": "/",
    "method": "post",
    "request": {
      "Content-type": "multipart/form-data",
      "data": {
        "type": "object",
        "properties": {
          "images": {
            "type": "file"
          }
        }
      }
    },
    "response": {
      "Content-type": "multipart/form-data",
      "data": {
        "type": "object",
        "properties": {
          "predicted_label": {
            "type": "string"
          },
          "scores": {
            "type": "array",
            "items": [{
              "type": "array",
              "minItems": 2,
              "maxItems": 2,
              "items": [{
                "type": "string"
              },
              {"type": "number"
            }]
          ]
        }
      }
    }
  }],
  "dependencies": [{
    "installer": "pip",
    "packages": [
```
Example of the Predictive Analytics Model Configuration File

The following code uses the TensorFlow engine as an example. You can modify the model_type parameter based on the actual engine type.

- Model input

```json
{
  "data": {
    "req_data": [
      {
        "buying_price": "high",
        "maint_price": "high",
        "doors": "2",
        "persons": "2",
        "lug_boot": "small",
        "safety": "low",
        "acceptability": "acc"
      },
      {
        "buying_price": "high",
        "maint_price": "high",
        "doors": "2",
        "persons": "2",
        "lug_boot": "small",
        "safety": "low",
        "acceptability": "acc"
      }
    ]
  }
}
```

- Model output

```json
{
  "data": {
    "resp_data": [
      {
        "predict_result": "unacc"
      },
      {
        "predict_result": "unacc"
      }
    ]
  }
}
```

- Configuration file

```json
{
  "model_type": "TensorFlow",
  "model_algorithm": "predict_analysis",
  "metrics": {
```
"f1": 0.345294,
"accuracy": 0.462963,
"precision": 0.338977,
"recall": 0.351852
},
"apis": [
{
"protocol": "http",
"url": "/",
"method": "post",
"request": {
"Content-type": "application/json",
"data": {
"type": "object",
"properties": {
"data": {
"type": "object",
"properties": {
"req_data": {
"items": [
{"type": "object",
"properties": {
}
]}
},
"type": "array"
}
}
},
"response": {
"Content-type": "multipart/form-data",
"data": {
"type": "object",
"properties": {
"data": {
"type": "object",
"properties": {
"resp_data": {
"type": "array",
"items": [
{"type": "object",
"properties": {
}
]}
]
}
}
}
}
},
"dependencies": [
{
"installer": "pip",
"packages": [
{
"restraint": "EXACT",
"package_version": "1.15.0",
"package_name": "numpy"
},
{
"restraint": "EXACT",
"package_version": "5.2.0",
"package_name": "Pillow"
]}
]
Example of the Custom Image Model Configuration File

The model input and output are similar to those in Example of the Object Detection Model Configuration File.

```json
{
    "model_algorithm": "image_classification",
    "model_type": "Image",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [{
        "protocol": "http",
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
                    "images": {
                        "type": "file"
                    }
                }
            }
        },
        "response": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "required": [
                    "predicted_label",
                    "scores"
                ],
                "properties": {
                    "predicted_label": {
                        "type": "string"
                    },
                    "scores": {
                        "type": "array",
                        "items": [
                            {"type": "array",
                             "minItems": 2,
                             "maxItems": 2,
                             "items": [{
                                "type": "string",
                                "type": "number"
                             }]
                          }]
                    }
                }
            }
        }
    },
    ...
}
}```
Example of the Machine Learning Model Configuration File

The following uses XGBoost as an example:

- **Model input**

  ```json
  {
    "data": {
      "req_data": [
        {
          "sepal_length": 5,
          "sepal_width": 3.3,
          "petal_length": 1.4,
          "petal_width": 0.2
        },
        {
          "sepal_length": 5,
          "sepal_width": 2,
          "petal_length": 3.5,
          "petal_width": 1
        },
        {
          "sepal_length": 6,
          "sepal_width": 2.2,
          "petal_length": 5,
          "petal_width": 1.5
        }
      ]
    }
  }
  ```

- **Model output**

  ```json
  {
    "data": {
      "resp_data": [
        {
          "predict_result": "Iris-setosa"
        },
        {
          "predict_result": "Iris-versicolor"
        }
      ]
    }
  }
  ```

- **Configuration file**

  ```json
  {
    "model_type": "XGBoost",
    "model_algorithm": "xgboost_iris_test",
    "runtime": "python2.7",
    "metrics": {
      "f1": 0.345294,
      "accuracy": 0.462963,
      "precision": 0.338977,
      "recall": 0.351852
    },
    "apis": [
      {
        "protocol": "http",
        "url": "/",
        "method": "post",
        "request": {
          "Content-type": "application/json",
          "data": {
            "type": "object",
            "properties": {
              "data": {
                "type": "object",
                "properties": {
                  "req_data": {
                    "type": "object",
                    "properties": {}
                  }
                }
              }
            }
          }
        }
      }
    ]
  }
  ```
Example of a Model Configuration File Using a Custom Dependency Package

The following example defines the NumPy 1.16.4 dependency environment.

```json
{
  "model_algorithm": "image_classification",
  "model_type": "TensorFlow",
  "runtime": "python3.6",
  "apis": [{
    "protocol": "http",
    "url": "/",
    "method": "post",
    "request": {
      "Content-type": "multipart/form-data",
      "data": {
        "type": "object",
        "properties": {
          "images": {
            "type": "file"
          }
        }
      }
    },
    "response": {
      "Content-type": "application/json",
      "data": {
        "type": "object",
        "properties": {
          "mnist_result": {
            "type": "array",
            "items": {
              "type": "string"
            }
          }
        }
      }
    }
  }
}
```
10.3 Specifications for Compiling Model Inference Code

This section describes how to compile model inference code in ModelArts. For details about the custom script code examples (including inference code examples) of common AI engines, see Examples of Custom Scripts. The following also provides an example of inference code for the TensorFlow engine and an example of customizing inference logic in an inference script.

Specifications for Compiling Inference Code

1. All custom Python code must be inherited from the BaseService class. Table 10-9 lists the import statements of different types of model parent classes.

Table 10-9 Import statements of the BaseService class

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Parent Class</th>
<th>Import Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>TfServingBaseService</td>
<td>from model_service.tfserving_model_service import TfServingBaseService</td>
</tr>
<tr>
<td>MXNet</td>
<td>MXNetBaseService</td>
<td>from mms.model_service.mxnet_model_service import MXNetBaseService</td>
</tr>
<tr>
<td>PyTorch</td>
<td>PTservingBaseService</td>
<td>from model_service.pytorch_model_service import PTservingBaseService</td>
</tr>
<tr>
<td>Pyspark</td>
<td>SparkServingBaseService</td>
<td>from model_service.spark_model_service import SparkServingBaseService</td>
</tr>
<tr>
<td>Caffe</td>
<td>CaffeBaseService</td>
<td>from model_service.caffe_model_service import CaffeBaseService</td>
</tr>
</tbody>
</table>
2. The following methods can be rewritten:

Table 10-10 Methods to be rewritten

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code> (self, model_name, model_path)</td>
<td>Initialization method, which is suitable for models created based on deep learning frameworks. Models and labels are loaded using this method. This method must be rewritten for models based on PyTorch and Caffe to implement the model loading logic.</td>
</tr>
<tr>
<td><code>__init__</code> (self, model_path)</td>
<td>Initialization method, which is suitable for models created based on machine learning frameworks. The model path (self.model_path) is initialized using this method. In Spark_MLlib, this method also initializes SparkSession (self.spark).</td>
</tr>
<tr>
<td><code>_preprocess</code>(self, data)</td>
<td>Preprocess method, which is called before an inference request and is used to convert the original request data of an API into the expected input data of a model</td>
</tr>
<tr>
<td><code>_inference</code>(self, data)</td>
<td>Inference request method. You are not advised to rewrite the method because once the method is rewritten, the built-in inference process of ModelArts will be overwritten and the custom inference logic will run.</td>
</tr>
<tr>
<td><code>_postprocess</code>(self, data)</td>
<td>Postprocess method, which is called after an inference request is complete and is used to convert the model output to the API output</td>
</tr>
</tbody>
</table>

**NOTE**
- You can choose to rewrite the preprocess and postprocess methods to implement preprocessing of the API input and postprocessing of the inference output.
- Rewriting the init method of the BaseService inheritance class may cause a model to run abnormally.

3. The attribute that can be used is the local path where the model resides. The attribute name is `self.model_path`. In addition, PySpark-based models can use `self.spark` to obtain the SparkSession object in `customize_service.py`.
NOTE

An absolute path is required for reading files in the inference code. You can obtain the absolute path of the model from the `self.model_path` attribute.

- When TensorFlow, Caffe, or MXNet is used, `self.model_path` indicates the path of the model file. See the following example:
  
  ```
  # Store the label.json file in the model directory. The following information is read:
  with open(os.path.join(self.model_path, 'label.json')) as f:
      self.label = json.load(f)
  ```

- When PyTorch, SciKit_Learn, or PySpark is used, `self.model_path` indicates the path of the model file. See the following example:
  
  ```
  # Store the label.json file in the model directory. The following information is read:
  dir_path = os.path.dirname(os.path.realpath(self.model_path))
  with open(os.path.join(dir_path, 'label.json')) as f:
      self.label = json.load(f)
  ```

4. Currently, two types of **content-type** APIs can be used for inputting data: `multipart/form-data` and `application/json`

- **multipart/form-data** request

  ```
  curl -X POST \n  <modelarts-inference-endpoint> \n  -F image1=@cat.jpg \n  -F images2=@horse.jpg
  ```

  The corresponding input data is as follows:

  ```
  [
  
  {  
    "image1":{
      "cat.jpg":"<cat..jpg file io>"
    }
  },
  
  {  
    "image2":{
      "horse.jpg":"<horse.jpg file io>"
    }
  }
  ]
  ```

- **application/json** request

  ```
  curl -X POST \n  <modelarts-inference-endpoint> \n  -d '{  
    "images":"base64 encode image"
  }'
  ```

  The corresponding input data is **python dict**.

  ```
  {
    "images":"base64 encode image"
  }
  ```

**TensorFlow Inference Script Example**

The following is an example of TensorFlow MnistService.

- **Inference code**

```python
from PIL import Image
import numpy as np
from model_service.tfserving_model_service import TfServingBaseService

class mnist_service(TfServingBaseService):
    def _preprocess(self, data):
        preprocessed_data = {}

        for k, v in data.items():
            # Preprocess data

        return preprocessed_data

if __name__ == '__main__':
    service = mnist_service()
    service.run()
```

ModelArts
User Guide (Senior AI Engineers)

Issue 01 (2021-01-15)  Copyright © Huawei Technologies Co., Ltd.  430
for file_name, file_content in v.items():
    image1 = Image.open(file_content)
    image1 = np.array(image1, dtype=np.float32)
    image1.resize((1, 784))
    preprocessed_data[k] = image1

return preprocessed_data

def _postprocess(self, data):
    infer_output = {}
    for output_name, result in data.items():
        infer_output["mnist_result"] = result[0].index(max(result[0]))
    return infer_output

● Request
  curl -X POST \ Real-time service address \ -F images=@test.jpg

● Response
  {"mnist_result": 7}

The preceding code example resizes images imported to the user's form to adapt to the model input shape. The \(32 \times 32\) image is read from the Pillow library and resized to \(1 \times 784\) to match the model input. In subsequent processing, convert the model output into a list for the RESTful API to display.

**XGBoost Inference Script Example**

```python
# coding:utf-8
import collections
import json
import xgboost as xgb
from model_service.python_model_service import XgSklServingBaseService

class user_Service(XgSklServingBaseService):
    # request data preprocess
    def _preprocess(self, data):
        list_data = []
        json_data = json.loads(data, object_pairs_hook=collections.OrderedDict)
        for element in json_data["data"]['req_data']:
            array = []
            for each in element:
                array.append(element[each])
            list_data.append(array)
        return list_data

    # predict
    def _inference(self, data):
        xg_model = xgb.Booster(model_file=self.model_path)
        pre_data = xgb.DMatrix(data)
        pre_result = xg_model.predict(pre_data)
        pre_result = pre_result.tolist()
        return pre_result

    # predict result process
    def _postprocess(self, data):
        resp_data = []
        for element in data:
            resp_data.append({"predict_result": element})
        return resp_data
```
Inference Script Example of the Custom Inference Logic

First, define a dependency package in the configuration file. For details, see Example of a Model Configuration File Using a Custom Dependency Package. Then, use the following code example to implement the loading and inference of the model in saved_model format.

```python
# -*- coding: utf-8 -*-
import json
import os
import threading
import numpy as np
import tensorflow as tf
from PIL import Image
from model_service.tfserving_model_service import TfServingBaseService
import logging
logger = logging.getLogger(__name__)

class MnistService(TfServingBaseService):
    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model_inputs = {}
        self.model_outputs = {}

        # The label file can be loaded here and used in the post-processing function.
        # Directories for storing the label.txt file on OBS and in the model package
        with open(os.path.join(self.model_path, 'label.txt')) as f:
            self.label = json.load(f)

        # Load the model in saved_model format in non-blocking mode to prevent blocking timeout.
        thread = threading.Thread(target=self.get_tf_sess)
        thread.start()

    def get_tf_sess(self):
        # Load the model in saved_model format.
        # The session will be reused. Do not use the with statement.
        sess = tf.Session(graph=tf.Graph())
        meta_graph_def = tf.saved_model.loader.load(sess, [tf.saved_model.tag_constants.SERVING], self.model_path)
        signature_defs = meta_graph_def.signature_def
        signature = []
        # only one signature allowed
        for signature_def in signature_defs:
            signature.append(signature_def)
            if len(signature) == 1:
                model_signature = signature[0]
            else:
                logger.warning("signatures more than one, use serving_default signature")
                model_signature = tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY
                logger.info("model signature: %s", model_signature)

        for signature_name in meta_graph_def[model_signature].inputs:
            tensorinfo = meta_graph_def.signature_def[model_signature].inputs[signature_name]
            name = tensorinfo.name
            op = self.sess.graph.get_tensor_by_name(name)
```

ModelArts
User Guide (Senior AI Engineers)
self.model_inputs[signature_name] = op
logger.info("model inputs: %s", self.model_inputs)

for signature_name in meta_graph_def.signature_def[model_signature].outputs:
tensorinfo = meta_graph_def.signature_def[model_signature].outputs[signature_name]
name = tensorinfo.name
op = self.sess.graph.get_tensor_by_name(name)
self.model_outputs[signature_name] = op
logger.info("model outputs: %s", self.model_outputs)

def _preprocess(self, data):
    # Two request modes using HTTPS
    # 1. The request in form-data file format is as follows: data = {
    #    "Request key value":{
    #        "File name":<File io>
    #    }
    # 2. Request in JSON format is as follows: data = json.loads("JSON body transferred by the API")
    preprocessed_data = {}
    for k, v in data.items():
        for file_name, file_content in v.items():
            image1 = Image.open(file_content)
            image1 = np.array(image1, dtype=np.float32)
            image1.resize((1, 28, 28))
            preprocessed_data[k] = image1
    return preprocessed_data

def _inference(self, data):
    feed_dict = {}
    for k, v in data.items():
        if k not in self.model_inputs.keys():
            logger.error("input key %s is not in model inputs %s", k, list(self.model_inputs.keys()))
            raise Exception("input key %s is not in model inputs %s" % (k, list(self.model_inputs.keys())))
        feed_dict[self.model_inputs[k]] = v
    result = self.sess.run(self.model_outputs, feed_dict=feed_dict)
    logger.info('predict result : ' + str(result))
    return result

def _postprocess(self, data):
    infer_output = {"mnist_result": []}
    for output_name, results in data.items():
        for result in results:
            infer_output["mnist_result"].append(np.argmax(result))
    return infer_output

def __del__(self):
    self.sess.close()
11.1 Model Template Overview

Because the configurations of models of the same function are similar, ModelArts integrates the configurations of such models into a common template. By using this template, you can easily and quickly import models without compiling the `config.json` configuration file. In simple terms, a template integrates AI engine and model configurations. Each template corresponds to a specific AI engine and inference mode. With the templates, you can quickly import models to ModelArts.

Background

Templates are classified into general and non-general templates.

- Non-general templates are customized for specific scenarios with the input and output mode fixed. For example, the **TensorFlow-based image classification template** uses the built-in image processing mode.
- General templates integrate a specific AI engine and running environment and use the undefined input and output mode. You need to select an input and output mode based on the model function or application scenario to overwrite the undefined mode. For example, an image classification model requires the built-in image processing mode, and an object detection model requires the built-in object detection mode.

**NOTE**

The models imported in undefined mode cannot be deployed as batch services.

Using a Template

The following uses the **TensorFlow-based image classification template** (For details, see [TensorFlow-py36 General Template](#)) as an example. You need to upload the TensorFlow model package to OBS in advance. Store the model files in the `model` directory. When creating a model using this template, you need to select the `model` directory.

1. On the **Import Model** page, set Meta Model Source to Template.
2. In the **Template** area, select **TensorFlow-based image classification template**.

ModelArts also provides three filter criteria: **Type**, **Engine**, and **Environment**, helping you quickly find the desired template. If the three filter criteria cannot meet your requirements, you can enter keywords to search for the target template.

**Figure 11-1 Selecting a template**

3. For **Model Folder**, select the **model** directory where the model files reside. For details, see **Template Description**.

**NOTE**

If a training job is executed for multiple times, different version directories are generated, such as V001 and V002, and the generated models are stored in the **model** folder in different version directories. When selecting model files, specify the **model** folder in the corresponding version directory.

4. If the default input and output mode of the selected template can be overwritten, you can select an input and output mode based on the model function or application scenario. **Input and Output Mode** is an abstract of the API in **config.json**. It describes the interface provided by the model for external inference. An input and output mode describes one or more APIs, and corresponds to a template.

For example, for **TensorFlow-based image classification template**, **Input and Output Mode** supports **Built-in image processing mode**. The input and output mode cannot be modified in the template. Therefore, you can only view but not modify the default input and output mode of the template on the page.

For details about the supported input and output modes, see **Input and Output Modes**.

**Supported Templates**

- **TensorFlow-Based Image Classification Template**
- **TensorFlow-py27 General Template**
- **TensorFlow-py36 General Template**
- **MXNet-py27 General Template**
- **MXNet-py36 General Template**
- **PyTorch-py27 General Template**
- **PyTorch-py36 General Template**
11.2 Template Description

11.2.1 TensorFlow-Based Image Classification Template

Introduction

AI engine: TensorFlow 1.8; Environment: Python 2.7. This template is used to import a TensorFlow-based image classification model saved in SavedModel format. This template uses the built-in image processing mode of ModelArts. For details about the image processing mode, see Built-in Image Processing Mode. Ensure that your model can process images whose key is images, because you need to input an image whose key is images to the model for inference. When using the template to import a model, select the model directory containing the model files.

Template Input

The template input is the TensorFlow-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see Model Package Example.

Input and Output Mode

Built-in Image Processing Mode cannot be overwritten. That is, you cannot select another input and output mode during model creation.

Model Package Specifications

- The model package must be stored in the OBS folder named model. Model files and the model inference code file are stored in the model folder.
- The model inference code file is optional. If the file exists, the file name must be customize_service.py. Only one inference code file can exist in the model folder. For details about how to compile the model inference code file, see Specifications for Compiling Model Inference Code.
- The structure of the model package imported using the template is as follows:
model/
   ├── Model file  //(Mandatory) The model file format varies according to the engine. For details, see the model package example.
   ├── Custom Python package  //(Optional) User's Python package, which can be directly referenced in the model inference code
   │    └── customize_service.py  //(Optional) Model inference code file. The file name must be customize_service.py. Otherwise, the code is not considered as inference code.

Model Package Example

Structure of the TensorFlow-based model package

When publishing the model, you only need to specify the model directory.

OBS bucket/directory name
   └── model  (Mandatory) The folder must be named model and is used to store model-related files.
      ├── <<Custom Python package>>  (Optional) User's Python package, which can be directly referenced in the model inference code
      │    └── saved_model.pb  (Mandatory) Protocol buffer file, which contains the diagram description of the model
      │         └── variables  Mandatory for the main file of the *.pb model. The folder must be named variables and contains the weight deviation of the model.
      │         │    ├── variables.index  Mandatory
      │         │    └── variables.data-00000-of-00001  Mandatory
      │    └── customize_service.py  (Optional) Model inference code file. The file must be named customize_service.py. Only one inference code file exists. The .py file on which customize_service.py depends can be directly put in the model directory.

11.2.2 TensorFlow-py27 General Template

Introduction

AI engine: TensorFlow 1.8; Environment: Python 2.7; Input and output mode: undefined mode. Select an appropriate input and output mode based on the model function or application scenario. When using the template to import a model, select the model directory containing the model files.

Template Input

The template input is the TensorFlow-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see Model Package Example.

Input and Output Mode

Undefined Mode can be overwritten. That is, you can select another input and output mode during model creation.

Model Package Specifications

- The model package must be stored in the OBS folder named model. Model files and the model inference code file are stored in the model folder.
- The model inference code file is optional. If the file exists, the file name must be customize_service.py. Only one inference code file can exist in the model folder. For details about how to compile the model inference code file, see Specifications for Compiling Model Inference Code.
- The structure of the model package imported using the template is as follows:
Model Package Example

Structure of the TensorFlow-based model package

When publishing the model, you only need to specify the model directory.

OBS bucket/directory name

- model (Mandatory) The folder must be named model and is used to store model-related files.
- <<Custom Python package>> (Optional) User's Python package, which can be directly referenced in the model inference code
- saved_model.pb (Mandatory) Protocol buffer file, which contains the diagram description of the model
- variables (Mandatory for the main file of the *.pb model. The folder must be named variables and contains the weight deviation of the model.
- variables.index (Mandatory)
- variables.data-00000-of-00001 (Mandatory)
- customize_service.py (Optional) Model inference code file. The file must be named customize_service.py. Only one inference code file exists. The .py file on which customize_service.py depends can be directly put in the model directory.

11.2.3 TensorFlow-py36 General Template

Introduction

AI engine: TensorFlow 1.8; Environment: Python 3.6; Input and output mode: undefined mode. Select an appropriate input and output mode based on the model function or application scenario. When using the template to import a model, select the model directory containing the model files.

Template Input

The template input is the TensorFlow-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see Model Package Example.

Input and Output Mode

Undefined Mode can be overwritten. That is, you can select another input and output mode during model creation.

Model Package Specifications

- The model package must be stored in the OBS folder named model. Model files and the model inference code file are stored in the model folder.
- The model inference code file is optional. If the file exists, the file name must be customize_service.py. Only one inference code file can exist in the model folder. For details about how to compile the model inference code file, see Specifications for Compiling Model Inference Code.
The structure of the model package imported using the template is as follows:

```plaintext
model/
    ├── Model file  // (Mandatory) The model file format varies according to the engine. For details, see the model package example.
    │    ├── Custom Python package  // (Optional) User's Python package, which can be directly referenced in the model inference code
    │    │    └── customize_service.py  // (Optional) Model inference code file. The file name must be `customize_service.py`. Otherwise, the code is not considered as inference code.
```

Model Package Example

**Structure of the TensorFlow-based model package**

When publishing the model, you only need to specify the `model` directory.

```plaintext
OBS bucket/directory name
├── model  // (Mandatory) The folder must be named `model` and is used to store model-related files.
│    ├── <<Custom Python package>>  // (Optional) User's Python package, which can be directly referenced in the model inference code
│    │    └── saved_model.pb  // (Mandatory) Protocol buffer file, which contains the diagram description of the model
│    │    └── variables  // Mandatory for the main file of the *.pb model. The folder must be named `variables` and contains the weight deviation of the model.
│    │        └── variables.index  // Mandatory
│    │        └── variables.data-00000-of-00001  // Mandatory
│    │    └── customize_service.py  // (Optional) Model inference code file. The file must be named `customize_service.py`. Only one inference code file exists. The `.py` file on which `customize_service.py` depends can be directly put in the `model` directory.
```

11.2.4 MXNet-py27 General Template

**Introduction**

AI engine: MXNet 1.2.1; Environment: Python 2.7; Input and output mode: undefined mode. Select an appropriate input and output mode based on the model function or application scenario. When using the template to import a model, select the `model` directory containing the model files.

**Template Input**

The template input is the MXNet-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see Model Package Example.

**Input and Output Mode**

**Undefined Mode** can be overwritten. That is, you can select another input and output mode during model creation.

**Model Package Specifications**

- The model package must be stored in the OBS folder named `model`. Model files and the model inference code file are stored in the `model` folder.
- The model inference code file is optional. If the file exists, the file name must be `customize_service.py`. Only one inference code file can exist in the `model` folder. For details about how to compile the model inference code file, see Specifications for Compiling Model Inference Code.
The structure of the model package imported using the template is as follows:

```
model/
    ├── Model file  //(Mandatory) The model file format varies according to the engine. For details, see the model package example.
    │  ├── Custom Python package  //(Optional) User's Python package, which can be directly referenced in the model inference code
    │  │  └── customize_service.py  //(Optional) Model inference code file. The file name must be `customize_service.py`. Otherwise, the code is not considered as inference code.
```

### Model Package Example

#### Structure of the MXNet-based model package

When publishing the model, you only need to specify the `model` directory.

```
OBS bucket/directory name
  ├── model  (Mandatory) The folder must be named `model` and is used to store model-related files. <<Custom Python package>>  (Optional) User’s Python package, which can be directly referenced in the model inference code
    │  └── resnet-50-symbol.json  (Mandatory) Model definition file, which contains the neural network description of the model
    │  └── resnet-50-0000.params  (Mandatory) Model variable parameter file, which contains parameter and weight information
    │  └── customize_service.py  (Optional) Model inference code file. The file must be named `customize_service.py`. Only one inference code file exists. The `.py` file on which `customize_service.py` depends can be directly put in the `model` directory.
```

### 11.2.5 MXNet-py36 General Template

#### Introduction

AI engine: MXNet 1.2.1; Environment: Python 3.6; Input and output mode: undefined mode. Select an appropriate input and output mode based on the model function or application scenario. When using the template to import a model, select the `model` directory containing the model files.

#### Template Input

The template input is the MXNet-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see [Model Package Example](#).

#### Input and Output Mode

**Undefined Mode** can be overwritten. That is, you can select another input and output mode during model creation.

#### Model Package Specifications

- The model package must be stored in the OBS folder named `model`. Model files and the model inference code file are stored in the `model` folder.
- The model inference code file is optional. If the file exists, the file name must be `customize_service.py`. Only one inference code file can exist in the `model` folder. For details about how to compile the model inference code file, see [Specifications for Compiling Model Inference Code](#).
The structure of the model package imported using the template is as follows:

```plaintext
template/
  ├── model/
  │     └── Model file
  │             //(Mandatory) The model file format varies according to the engine. For details, see the model package example.
  │     └── Custom Python package
  │             //(Optional) User's Python package, which can be directly referenced in the model inference code
  │     └── customize_service.py
  │             //(Optional) Model inference code file. The file name must be customize_service.py. Otherwise, the code is not considered as inference code.
```

**Model Package Example**

**Structure of the MXNet-based model package**

When publishing the model, you only need to specify the `model` directory.

```
OBS bucket/directory name
├── model    (Mandatory) The folder must be named `model` and is used to store model-related files.
├── <<Custom Python package>>    (Optional) User's Python package, which can be directly referenced in the model inference code
├── resnet-50-symbol.json    (Mandatory) Model definition file, which contains the neural network description of the model
├── resnet-50-0000.params    (Mandatory) Model variable parameter file, which contains parameter and weight information
├── customize_service.py    (Optional) Model inference code file. The file must be named `customize_service.py`. Only one inference code file exists. The `.py` file on which `customize_service.py` depends can be directly put in the `model` directory.
```

### 11.2.6 PyTorch-py27 General Template

**Introduction**

AI engine: PyTorch 1.0; Environment: Python 2.7; Input and output mode: undefined mode. Select an appropriate input and output mode based on the model function or application scenario. When using the template to import a model, select the `model` directory containing the model files.

**Template Input**

The template input is the PyTorch-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see Model Package Example.

**Input and Output Mode**

*Undefined Mode* can be overwritten. That is, you can select another input and output mode during model creation.

**Model Package Specifications**

- The model package must be stored in the OBS folder named `model`. Model files and the model inference code file are stored in the `model` folder.

- The model inference code file is optional. If the file exists, the file name must be `customize_service.py`. Only one inference code file can exist in the `model` folder. For details about how to compile the model inference code file, see Specifications for Compiling Model Inference Code.

- The structure of the model package imported using the template is as follows:
Model Package Example

Structure of the PyTorch-based model package

When publishing the model, you only need to specify the model directory.

```plaintext
OBS bucket/directory name
├── model (Mandatory) The folder must be named model and is used to store model-related files.
│   └── <<Custom Python package>> (Optional) User's Python package, which can be directly referenced in the model inference code
│       └── resnet50.pth (Mandatory) PyTorch model file, which contains variable and weight information
│           └── customize_service.py (Optional) Model inference code file. The file name must be customize_service.py. Only one inference code file exists. The .py file on which customize_service.py depends can be directly put in the model directory.
```

11.2.7 PyTorch-py36 General Template

Introduction

AI engine: PyTorch 1.0; Environment: Python 3.6; Input and output mode: undefined mode. Select an appropriate input and output mode based on the model function or application scenario. When using the template to import a model, select the model directory containing the model files.

Template Input

The template input is the PyTorch-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see Model Package Example.

Input and Output Mode

Undefined Mode can be overwritten. That is, you can select another input and output mode during model creation.

Model Package Specifications

- The model package must be stored in the OBS folder named model. Model files and the model inference code file are stored in the model folder.
- The model inference code file is optional. If the file exists, the file name must be customize_service.py. Only one inference code file can exist in the model folder. For details about how to compile the model inference code file, see Specifications for Compiling Model Inference Code.
- The structure of the model package imported using the template is as follows:

```plaintext
model/
│   └── Model file (Mandatory) The model file format varies according to the engine. For details, see the model package example.
│   └── Custom Python package (Optional) User's Python package, which can be directly referenced in the model inference code
│       └── customize_service.py (Optional) Model inference code file. The file name must be customize_service.py. Otherwise, the code is not considered as inference code.
```
Model Package Example

Structure of the PyTorch-based model package

When publishing the model, you only need to specify the model directory.

11.2.8 Caffe-CPU-py27 General Template

Introduction

AI engine: CPU-based Caffe 1.0; Environment: Python 2.7; Input and output mode: undefined mode. Select an appropriate input and output mode based on the model function or application scenario. When using the template to import a model, select the model directory containing the model files.

Template Input

The template input is the Caffe-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see Model Package Example.

Input and Output Mode

Undefined Mode can be overwritten. That is, you can select another input and output mode during model creation.

Model Package Specifications

- The model package must be stored in the OBS folder named model. Model files and the model inference code file are stored in the model folder.
- The model inference code file is optional. If the file exists, the file name must be customize_service.py. Only one inference code file can exist in the model folder. For details about how to compile the model inference code file, see Specifications for Compiling Model Inference Code.
- The structure of the model package imported using the template is as follows:

```plaintext
model/
    ├── Model file  //(Mandatory) The model file format varies according to the engine. For details, see the model package example.
    │      ├── Model inference code file referenced in the model inference code
    │      ├── Custom Python package  //(Optional) User’s Python package, which can be directly referenced in the model inference code
    │      └── customize_service.py //(Optional) Model inference code file. The file name must be customize_service.py. Otherwise, the code is not considered as inference code.
```
## Model Package Example

### Structure of the Caffe-based model package

When publishing the model, you only need to specify the **model** directory.

<table>
<thead>
<tr>
<th>OBS bucket/directory name</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>├── model</td>
<td>(Mandatory) The folder must be named <strong>model</strong> and is used to store model-related files.</td>
<td></td>
</tr>
<tr>
<td>│   └── &lt;&lt;Custom Python package&gt;&gt;</td>
<td>(Optional) User’s Python package, which can be directly referenced in the model inference code</td>
<td></td>
</tr>
<tr>
<td>│   └── deploy.prototxt</td>
<td>(Mandatory) Caffe model file, which contains information such as the model network structure</td>
<td></td>
</tr>
<tr>
<td>│   └── resnet.caffemodel</td>
<td>(Mandatory) Caffe model file, which contains variable and weight information</td>
<td></td>
</tr>
<tr>
<td>│   └── customize_service.py</td>
<td>(Optional) Model inference code file. The file name must be <code>customize_service.py</code>. Only one inference code file exists. The <code>.py</code> file on which <code>customize_service.py</code> depends can be directly put in the <strong>model</strong> directory.</td>
<td></td>
</tr>
</tbody>
</table>

### 11.2.9 Caffe-GPU-py27 General Template

#### Introduction

AI engine: GPU-based Caffe 1.0; Environment: Python 2.7; Input and output mode: undefined mode. Select an appropriate input and output mode based on the model function or application scenario. When using the template to import a model, select the **model** directory containing the model files.

#### Template Input

The template input is the Caffe-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see **Model Package Example**.

#### Input and Output Mode

**Undefined Mode** can be overwritten. That is, you can select another input and output mode during model creation.

#### Model Package Specifications

- The model package must be stored in the OBS folder named **model**. Model files and the model inference code file are stored in the **model** folder.

- The model inference code file is optional. If the file exists, the file name must be `customize_service.py`. Only one inference code file can exist in the **model** folder. For details about how to compile the model inference code file, see **Specifications for Compiling Model Inference Code**.

- The structure of the model package imported using the template is as follows:

  ```
  model/
  ├── Model file     // (Mandatory) The model file format varies according to the engine. For details, see the model package example.
  ├── Custom Python package // (Optional) User’s Python package, which can be directly referenced in the model inference code
  └── customize_service.py     // (Optional) Model inference code file. The file name must be `customize_service.py`. Otherwise, the code is not considered as inference code.
  ```

### Model Package Example

#### Structure of the Caffe-based model package
When publishing the model, you only need to specify the **model** directory.

**OBS bucket/directory name**
- model (Mandatory) The folder must be named **model** and is used to store model-related files.
- <<Custom Python package>> (Optional) User's Python package, which can be directly referenced in the model inference code
- deploy.prototxt (Mandatory) Caffe model file, which contains information such as the model network structure
- resnet.caffemodel (Mandatory) Caffe model file, which contains variable and weight information
- customize_service.py (Optional) Model inference code file. The file must be named **customize_service.py**. Only one inference code file exists. The .py file on which **customize_service.py** depends can be directly put in the **model** directory.

### 11.2.10 Caffe-CPU-py36 General Template

**Introduction**

AI engine: CPU-based Caffe 1.0; Environment: Python 3.6; Input and output mode: undefined mode. Select an appropriate input and output mode based on the model function or application scenario. When using the template to import a model, select the **model** directory containing the model files.

**Template Input**

The template input is the Caffe-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see **Model Package Example**.

**Input and Output Mode**

**Undefined Mode** can be overwritten. That is, you can select another input and output mode during model creation.

**Model Package Specifications**

- The model package must be stored in the OBS folder named **model**. Model files and the model inference code file are stored in the **model** folder.
- The model inference code file is optional. If the file exists, the file name must be **customize_service.py**. Only one inference code file can exist in the **model** folder. For details about how to compile the model inference code file, see **Specifications for Compiling Model Inference Code**.
- The structure of the model package imported using the template is as follows:

  ```
  model/
  ├── Model file  //(Mandatory) The model file format varies according to the engine. For details, see the model package example.
  └── Custom Python package  //(Optional) User's Python package, which can be directly referenced in the model inference code
    └── customize_service.py  //(Optional) Model inference code file. The file name must be **customize_service.py**. Otherwise, the code is not considered as inference code.
  ```

**Model Package Example**

**Structure of the Caffe-based model package**

When publishing the model, you only need to specify the **model** directory.
11.2.11 Caffe-GPU-py36 General Template

Introduction

AI engine: GPU-based Caffe 1.0; Environment: Python 3.6; Input and output mode: undefined mode. Select an appropriate input and output mode based on the model function or application scenario. When using the template to import a model, select the model directory containing the model files.

Template Input

The template input is the Caffe-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see Model Package Example.

Input and Output Mode

Undefined Mode can be overwritten. That is, you can select another input and output mode during model creation.

Model Package Specifications

- The model package must be stored in the OBS folder named model. Model files and the model inference code file are stored in the model folder.
- The model inference code file is optional. If the file exists, the file name must be customize_service.py. Only one inference code file can exist in the model folder. For details about how to compile the model inference code file, see Specifications for Compiling Model Inference Code.
- The structure of the model package imported using the template is as follows:

```
├── model    (Mandatory) The folder must be named model and is used to store model-related files.
│    ├── <<Custom Python package>>      (Optional) User's Python package, which can be directly referenced in the model inference code
│    │   │   – deploy.prototxt         (Mandatory) Caffe model file, which contains information such as the model network structure
│    │   │   – resnet.caffemodel     (Mandatory) Caffe model file, which contains variable and weight information
│    │   │   – customize_service.py  (Optional) Model inference code file. The file must be named customize_service.py. Only one inference code file exists. The .py file on which customize_service.py depends can be directly put in the model directory.
│    │   └── customize_service.py  (Optional) Model inference code file. The file name must be customize_service.py. Otherwise, the code is not considered as inference code.
```

Model Package Example

Structure of the Caffe-based model package

When publishing the model, you only need to specify the model directory.

```
├── model    (Mandatory) The folder must be named model and is used to store model-related files.
│    ├── <<Custom Python package>>      (Optional) User's Python package, which can be directly referenced in the model inference code
│    │   │   – deploy.prototxt         (Mandatory) Caffe model file, which contains information such as the model network structure
```

11.2.12 ARM-Ascend Template

Introduction

AI engine: MindSpore; Environment: Python 3.5; Input and output mode: undefined mode. Select an appropriate input and output mode based on the model function or application scenario. When using the template to import a model, select the model directory containing the model files.

Template Input

The template input is the OM-based model package stored on OBS. Ensure that the OBS directory you use and ModelArts are in the same region. For details about model package requirements, see Model Package Example.

Input and Output Mode

Undefined Mode can be overwritten. That is, you cannot select another input and output mode during model creation.

Model Package Specifications

- The model package must be stored in the OBS folder named model. Model files and the model inference code file are stored in the model folder.
- The model inference code file is optional. If the file exists, the file name must be customize_service.py. Only one inference code file can exist in the model folder. For details about how to compile the model inference code file, see Specifications for Compiling Model Inference Code.
- The structure of the model package imported using the template is as follows:

```
model/
├── Model file  //(Mandatory) The model file format varies according to the engine. For details, see the model package example.
├── Custom Python package  //(Optional) User's Python package, which can be directly referenced in the model inference code
└── customize_service.py  //(Optional) Model inference code file. The file name must be customize_service.py. Otherwise, the code is not considered as inference code.
```

Model Package Example

Structure of the OM-based model package

When publishing the model, you only need to specify the model directory.
11.3 Input and Output Modes

11.3.1 Built-in Object Detection Mode

Input

This is a built-in input and output mode for object detection. The models using this mode are identified as object detection models. The prediction request path is /, the request protocol is HTTP, the request method is POST, Content-Type is multipart/form-data, key is images, and type is file. Before selecting this mode, ensure that your model can process the input data whose key is images.

Output

The inference result is returned in JSON format. For details about the fields, see Table 11-1.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>detection_classes</td>
<td>String array</td>
<td>List of detected objects, for example, [&quot;yunbao&quot;,&quot;cat&quot;]</td>
</tr>
<tr>
<td>detection_boxes</td>
<td>Float array</td>
<td>Coordinates of the bounding box, in the format of $[Y_{min}, X_{min}, Y_{max}, X_{max}]$.</td>
</tr>
<tr>
<td>detection_scores</td>
<td>Float array</td>
<td>Confidence scores of detected objects, which are used to measure the detection accuracy</td>
</tr>
</tbody>
</table>

The JSON Schema of the inference result is as follows:

```json
{  "type": "object",  "properties": {    "detection_classes": {      "items": {        "type": "string"      },      "type": "array"    },    "detection_boxes": {      "items": {        "minItems": 4,        "items": {          "type": "number"        }      },      "type": "array"    }  }}
```
Sample Request

In this mode, input an image to be processed in the inference request. The inference result is returned in JSON format. The following are examples:

- Performing prediction on the console
  On the Prediction tab page of the service details page, upload an image and click Predict to obtain the prediction result.

- Using Postman to call a RESTful API for prediction
  After a model is deployed as a service, you can obtain the API URL on the Usage Guides tab page of the service details page.
  - On the Headers tab page, set Content-Type to multipart/form-data and X-Auth-Token to the actual token obtained.
  
  ![Figure 11-2 Setting the request header](image)

  - On the Body tab page, set the request body. Set key to images, select File, select the image to be processed, and click send to send your prediction request.
  
  ![Figure 11-3 Setting the request body](image)
11.3.2 Built-in Image Processing Mode

Input

The built-in image processing input and output mode can be applied to models such as image classification, object detection, and image semantic segmentation. The prediction request path is /, the request protocol is HTTPS, the request method is POST, Content-Type is multipart/form-data, key is images, and type is file. Before selecting this mode, ensure that your model can process the input data whose key is images.

Output

The inference result is returned in JSON format. The specific fields are determined by the model.

Sample Request

In this mode, input an image to be processed in the inference request. The response in JSON format varies according to the model. The following are examples:

- Performing prediction on the console
- Using Postman to call a RESTful API for prediction
  
  After a model is deployed as a service, you can obtain the API URL on the Usage Guides tab page of the service details page. On the Body tab page, set the request body. Set key to images, select File, select the image to be processed, and click send to send your prediction request.

![Calling a RESTful API](image)

11.3.3 Built-in Predictive Analytics Mode

Input

This is a built-in input and output mode for predictive analytics. The models using this mode are identified as predictive analytics models. The prediction request path is /, the request protocol is HTTP, the request method is POST, and Content-Type is application/json. The request body is in JSON format. For details about the JSON fields, see Table 11-2. Before selecting this mode, ensure that your model can process the input data in JSON Schema format.
### Table 11-2 JSON field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Data structure</td>
<td>Inference data. For details, see Table 11-3.</td>
</tr>
</tbody>
</table>

### Table 11-3 Data description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_data</td>
<td>ReqData array</td>
<td>List of inference data</td>
</tr>
</tbody>
</table>

**ReqData** is of the **Object** type and indicates the inference data. The data structure is determined by the application scenario. For models using this mode, the preprocessing logic in the custom model inference code should be able to correctly process the data inputted in the format defined by the mode.

The **JSON Schema** of a prediction request is as follows:

```json
{
  "type": "object",
  "properties": {
    "data": {
      "type": "object",
      "properties": {
        "req_data": {
          "items": [{}],
          "type": "array"
        }
      }
    }
  }
}
```

### Output

The inference result is returned in JSON format. For details about the JSON fields, see **Table 11-4**.

### Table 11-4 JSON field description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Data structure</td>
<td>Inference data. For details, see Table 11-5.</td>
</tr>
</tbody>
</table>
Table 11-5 Data description

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resp_data</td>
<td>RespData array</td>
<td>List of prediction results</td>
</tr>
</tbody>
</table>

Similar to ReqData, RespData is also of the Object type and indicates the prediction result. Its structure is determined by the application scenario. For models using this mode, the postprocessing logic in the custom model inference code should be able to correctly output data in the format defined by the mode.

The JSON Schema of a prediction result is as follows:

```
{
  "type": "object",
  "properties": {
    "data": {
      "type": "object",
      "properties": {
        "resp_data": {
          "type": "array",
          "items": [{
            "type": "object",
            "properties": {}
          }]
        }
      }
    }
  }
}
```

Sample Request

In this mode, input the data to be predicted in JSON format. The prediction result is returned in JSON format. The following are examples:

- Performing prediction on the console
  On the Prediction tab page of the service details page, enter inference code and click Predict to obtain the prediction result.

- Using Postman to call a RESTful API for prediction
  After a model is deployed as a service, you can obtain the API URL on the Usage Guides tab page of the service details page.
  - On the Headers tab page, set Content-Type to application/json and X-Auth-Token to the actual token obtained.
  - On the Body tab page, edit the data to be predicted and click send to send your prediction request.

Figure 11-5 Setting the request header for prediction

<table>
<thead>
<tr>
<th>Headers</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-Type</td>
<td>application/json</td>
<td></td>
</tr>
<tr>
<td>X-Auth-Token</td>
<td>[token]</td>
<td></td>
</tr>
</tbody>
</table>
11.3.4 Undefined Mode

Description

The undefined mode does not define the input and output mode. The input and output mode is determined by the model. Select this mode only when the existing input and output mode is not applicable to the application scenario of the model. The models imported in undefined mode cannot be deployed as batch services. In addition, the service prediction page may not be displayed properly. New modes are coming soon for more application scenarios.

Input

No limit.

Output

No limit.

Sample Request

The undefined mode has no specific sample request because the input and output of the request are entirely determined by the model.
12 Examples of Custom Scripts

## 12.1 TensorFlow

TensorFlow has two types of APIs: Keras and tf. Keras and tf use different code for training and saving models, but the same code for inference.

### Training a Model (Keras API)

```python
from keras.models import Sequential
model = Sequential()
from keras.layers import Dense
import tensorflow as tf

# Import a training dataset.
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
print(x_train.shape)

from keras.layers import Dense
from keras.models import Sequential
import keras
from keras.layers import Dense, Activation, Flatten, Dropout

# Define a model network.
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(units=5120, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(units=10, activation='softmax'))

# Define an optimizer and loss functions.
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
# Train the model.
model.fit(x_train, y_train, epochs=2)
# Evaluate the model.
model.evaluate(x_test, y_test)
```
Saving a Model (Keras API)

```python
from keras import backend as K

# K.get_session().run(tf.global_variables_initializer())

# Define the inputs and outputs of the prediction API.
# The key values of the inputs and outputs dictionaries are used as the index keys for the input and output
tensors of the model.
# The input and output definitions of the model must match the custom inference script.
predict_signature = tf.saved_model.signature_def_utils.predict_signature_def(
    inputs={"images" : model.input},
    outputs={"scores" : model.output}
)

# Define a save path.
builder = tf.saved_model.builder.SavedModelBuilder('./mnist_keras/)

builder.add_meta_graph_and_variables(
    sess = K.get_session(),
    # The tf.saved_model.tag_constants.SERVING tag needs to be defined for inference and deployment.
    tags=[tf.saved_model.tag_constants.SERVING],
    # signature_def_map: Only single items can exist, or the corresponding key needs to be defined as follows:
    # tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY
    signature_def_map={
        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY: predict_signature
    }
)

builder.save()
```

Training a Model (tf API)

```python
from __future__ import print_function
import gzip
import os
import urllib
import numpy
import tensorflow as tf
from six.moves import urllib

# Training data is obtained from the Yann LeCun official website http://yann.lecun.com/exdb/mnist/.
SOURCE_URL = 'http://yann.lecun.com/exdb/mnist/
TRAIN_IMAGES = 'train-images-idx3-ubyte.gz'
TRAIN_LABELS = 'train-labels-idx1-ubyte.gz'
TEST_IMAGES = 't10k-images-idx3-ubyte.gz'
TEST_LABELS = 't10k-labels-idx1-ubyte.gz'
VALIDATION_SIZE = 5000

def maybe_download(filename, work_directory):
    # Download the data from Yann's website, unless it's already here.
    if not os.path.exists(work_directory):
        os.mkdir(work_directory)
    filepath = os.path.join(work_directory,
        filename)
    if not os.path.exists(filepath):
        filepath, _ = urllib.request.urlretrieve(SOURCE_URL + filename, filepath)
        statinfo = os.stat(filepath)
        print('Successfully downloaded %s %d bytes.' % (filename, statinfo.st_size))
    return filepath

def _read32(bytestream):
    # Read an array of 32 bits from bytestream
    order = np.dtype(np.int32).itemsize * 8
    dtype = np.dtype(np.int32).newbyteorder('>')
    return np.frombuffer(bytestream.read(order), dtype=dtype)[0]
```
```python
dt = numpy.dtype(numpy.uint32).newbyteorder('>').
return numpy.frombuffer(bytestream.read(4), dtype=dt)[0]

def extract_images(filename):
    """Extract the images into a 4D uint8 numpy array [index, y, x, depth]."""
    print('Extracting %s' % filename)
    with gzip.open(filename) as bytestream:
        magic = _read32(bytestream)
        if magic != 2051:
            raise ValueError(  
                'Invalid magic number %d in MNIST image file: %s' %  
                (magic, filename))
        num_images = _read32(bytestream)
        rows = _read32(bytestream)
        cols = _read32(bytestream)
        buf = bytestream.read(rows * cols * num_images)
        data = numpy.frombuffer(buf, dtype=numpy.uint8)
        data = data.reshape(num_images, rows, cols, 1)
    return data

def dense_to_one_hot(labels_dense, num_classes=10):
    """Convert class labels from scalars to one-hot vectors."""
    num_labels = labels_dense.shape[0]
    index_offset = numpy.arange(num_labels) * num_classes
    labels_one_hot = numpy.zeros((num_labels, num_classes))
    labels_one_hot.flat[index_offset + labels_dense.ravel()] = 1
    return labels_one_hot

def extract_labels(filename, one_hot=False):
    """Extract the labels into a 1D uint8 numpy array [index]."""
    print('Extracting %s' % filename)
    with gzip.open(filename) as bytestream:
        magic = _read32(bytestream)
        if magic != 2049:
            raise ValueError(  
                'Invalid magic number %d in MNIST label file: %s' %  
                (magic, filename))
        num_items = _read32(bytestream)
        buf = bytestream.read(num_items)
        labels = numpy.frombuffer(buf, dtype=numpy.uint8)
        if one_hot:
            return dense_to_one_hot(labels)
        return labels

class DataSet(object):
    """Class encompassing test, validation and training MNIST data set."""
    def __init__(self, images, labels, fake_data=False, one_hot=False):
        """Construct a DataSet. one_hot arg is used only if fake_data is True."""
        if fake_data:
            self._num_examples = 10000
            self.one_hot = one_hot
        else:
            assert images.shape[0] == labels.shape[0], (  
                'images.shape: %s labels.shape: %s' % (images.shape,  
                labels.shape))
            self._num_examples = images.shape[0]
            self._num_examples = images.shape[0]

        # Convert shape from [num examples, rows, columns, depth]
        # to [num examples, rows*columns] (assuming depth == 1)
        assert images.shape[3] == 1
        images = images.reshape(images.shape[0],  
                images.shape[1] * images.shape[2])

        # Convert from [0, 255] -> [0.0, 1.0].
```

images = images.astype(numpy.float32)
images = numpy.multiply(images, 1.0 / 255.0)
self._images = images
self._labels = labels
self._epochs_completed = 0
self._index_in_epoch = 0

@property
def images(self):
    return self._images

@property
def labels(self):
    return self._labels

@property
def num_examples(self):
    return self._num_examples

@property
def epochs_completed(self):
    return self._epochs_completed

def next_batch(self, batch_size, fake_data=False):
    """Return the next 'batch_size' examples from this data set.""
    if fake_data:
        fake_image = [1] * 784
        if self.one_hot:
            fake_label = [1] + [0] * 9
        else:
            fake_label = 0
        return [fake_image for _ in range(batch_size)], [
            fake_label for _ in range(batch_size)
        ]
    start = self._index_in_epoch
    self._index_in_epoch += batch_size
    if self._index_in_epoch > self._num_examples:
        # Finished epoch
        self._epochs_completed += 1
        # Shuffle the data
        perm = numpy.arange(self._num_examples)
        numpy.random.shuffle(perm)
        self._images = self._images[perm]
        self._labels = self._labels[perm]
        # Start next epoch
        start = 0
    self._index_in_epoch = batch_size
    assert batch_size <= self._num_examples
    end = self._index_in_epoch
    return self._images[start:end], self._labels[start:end]

def read_data_sets(train_dir, fake_data=False, one_hot=False):
    """Return training, validation and testing data sets.""
    class DataSets(object):
        pass

data_sets = DataSets()

    if fake_data:
        data_sets.train = DataSet([], [], fake_data=True, one_hot=one_hot)
        data_sets.validation = DataSet([], [], fake_data=True, one_hot=one_hot)
        data_sets.test = DataSet([], [], fake_data=True, one_hot=one_hot)
        return data_sets

local_file = maybe_download(TRAIN.Images, train_dir)
train_images = extract_images(local_file)
local_file = maybe_download(TRAIN_LABELS, train_dir)
train_labels = extract_labels(local_file, one_hot=one_hot)

local_file = maybe_download(TEST_IMAGES, train_dir)
test_images = extract_images(local_file)

local_file = maybe_download(TEST_LABELS, train_dir)
test_labels = extract_labels(local_file, one_hot=one_hot)

validation_images = train_images[:VALIDATION_SIZE]
validation_labels = train_labels[:VALIDATION_SIZE]
train_images = train_images[VALIDATION_SIZE:]
train_labels = train_labels[VALIDATION_SIZE:]

data_sets.train = DataSet(train_images, train_labels)
data_sets.validation = DataSet(validation_images, validation_labels)
data_sets.test = DataSet(test_images, test_labels)
return data_sets

training_iteration = 1000

modelarts_example_path = './modelarts-mnist-train-save-deploy-example'
export_path = modelarts_example_path + '/model/
data_path = './'

print('Training model...')
mnist = read_data_sets(data_path, one_hot=True)
sess = tf.InteractiveSession()

serialized_tf_example = tf.placeholder(tf.string, name='tf_example')
feature_configs = {'x': tf.FixedLenFeature(shape=[784], dtype=tf.float32),}

feature_configs = {'x': tf.FixedLenFeature(shape=[784], dtype=tf.float32),}

x = tf.placeholder('float', shape=[None, 10])

w = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
sess.run(tf.global_variables_initializer())
y = tf.nn.softmax(tf.matmul(x, w) + b, name='y')

print('Done training!')

# Export the model.
# The model needs to be saved using the saved_model API.
print('Exporting trained model to', export_path)
builder = tf.saved_model.builder.SavedModelBuilder(export_path)
tensor_info_x = tf.saved_model.utils.build_tensor_info(x)
tensor_info_y = tf.saved_model.utils.build_tensor_info(y)

# Define the inputs and outputs of the prediction API.
# The key values of the inputs and outputs dictionaries are used as the index keys for the input and output tensors of the model.
# The input and output definitions of the model must match the custom inference script.

prediction_signature = (tf.saved_model.signature_def_utils.build_signature_def(
    inputs={'images': tensor_info_x},
    outputs={'scores': tensor_info_y},
    method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME))

legacy_init_op = tf.group(tf.tables_initializer(), name='legacy_init_op')
built.add_meta_graph_and_variables(
    # Set tag to serve/tf.saved_model.tag_constants.SERVING.
    sess, [tf.saved_model.tag_constants.SERVING],
    signature_def_map={'predict_images': prediction_signature},
    legacy_init_op=legacy_init_op)
built.save()
print('Done exporting!')

Inference Code (Keras and tf APIs)

from PIL import Image
import numpy as np
from model_service.tfserving_model_service import TfServingBaseService

class mnist_service(TfServingBaseService):
    # Match the model input with the user's HTTPS API input during preprocessing.
    # The model input corresponding to the preceding training part is {'images':<array>}.  
    def _preprocess(self, data):
        preprocessed_data = {}
        images = []
        # Iterate the input data.
        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((1,784))
                images.append(image1)
        # Return the numpy array.
        images = np.array(images,dtype=np.float32)
        # Perform batch processing on multiple input samples and ensure that the shape is the same as that
        # inputted during training.
        images.resize((len(data), 784))
        preprocessed_data['images'] = images
        return preprocessed_data

    # Processing logic of the inference for invoking the parent class.

    # The output corresponding to model saving in the preceding training part is {'scores':<array>}.  
    # Postprocess the HTTPS output.
    def _postprocess(self, data):
        infer_output = {'mnist_result': []}
        # Iterate the model output.
        for output_name, results in data.items():
            for result in results:
                infer_output['mnist_result'].append(result.index(max(result)))
        return infer_output
12.2 TensorFlow 2.1

Training and Saving a Model

```python
from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28, 28)),
                                      tf.keras.layers.Dense(128, activation='relu'),
                                      tf.keras.layers.Dense(256, activation='relu'),
                                      tf.keras.layers.Dropout(0.2),
                                      # Name the output layer output, which is used to obtain the result during model inference.
                                      tf.keras.layers.Dense(10, activation='softmax', name="output")])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)
tf.keras.models.save_model(model, ".\mnist")
```

Inference Code

```python
import logging
import threading
import numpy as np
import tensorflow as tf
from PIL import Image
from model_service.tfserving_model_service import TfServingBaseService

logger = logging.getLogger()
logger.setLevel(logging.INFO)

class mnist_service(TfServingBaseService):
    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model = None
        self.predict = None

        # The label file can be loaded here and used in the post-processing function.
        # Directories for storing the label.txt file on OBS and in the model package

        with open(os.path.join(self.model_path, 'label.txt')) as f:
            self.label = json.load(f)

        # Load the model in saved_model format in non-blocking mode to prevent blocking timeout.
        thread = threading.Thread(target=self.load_model)
        thread.start()

    def load_model(self):
        # Load the model in saved_model format.
        self.model = tf.saved_model.load(self.model_path)
```

signature defs = self.model.signatures.keys()

signature = []
# only one signature allowed
for signature_def in signature_defs:
    signature.append(signature_def)

if len(signature) == 1:
    model_signature = signature[0]
else:
    logging.warning("signatures more than one, use serving_default signature from \%s", signature)
    model_signature = tf.saved_model.DEFAULT_SERVING_SIGNATURE_DEF_KEY

self.predict = self.model.signatures[model_signature]

def _preprocess(self, data):
    images = []
    for k, v in data.items():
        for file_name, file_content in v.items():
            image1 = Image.open(file_content)
            image1 = np.array(image1, dtype=np.float32)
            image1.resize((28, 28, 1))
            images.append(image1)
    images = tf.convert_to_tensor(images, dtype=tf.dtypes.float32)
    preprocessed_data = images
    return preprocessed_data

def _inference(self, data):
    return self.predict(data)

def _postprocess(self, data):
    return {
        "result": int(data["output"].numpy()[0].argmax())
    }

12.3 PyTorch

Training a Model

from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms

# Define a network structure.
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # The second dimension of the input must be 784.
        self.hidden1 = nn.Linear(784, 5120, bias=False)
        self.output = nn.Linear(5120, 10, bias=False)

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = F.relu(self.hidden1(x))
        x = F.dropout(x, 0.2)
        x = self.output(x)
        return F.log_softmax(x)
def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]	Loss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))

def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    print('
Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)
'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

device = torch.device("cpu")
batch_size=64
kwargs={}  
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('.', train=True, download=True,
                   transform=transforms.Compose([transforms.ToTensor()]),
                   batch_size=batch_size, shuffle=True, **kwargs),
    batch_size=batch_size, shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('.', train=False, transform=transforms.Compose([transforms.ToTensor()]),
                   batch_size=1000, shuffle=True, **kwargs),
    batch_size=1000, shuffle=True, **kwargs)

model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
optimizer = optim.Adam(model.parameters())

for epoch in range(1, 2 + 1):
    train(model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)

# Saving a Model

# The model must be saved using state_dict and can be deployed remotely.
torch.save(model.state_dict(), "pytorch_mnist/mnist_mlp.pt")

# Inference Code

from PIL import Image
import log
from model_service.pytorch_model_service import PTServingBaseService
import torch.nn.functional as F
import torch.nn as nn
import torch
import json
import numpy as np
logger = log.getLogger(__name__)
import torchvision.transforms as transforms

# Define model preprocessing.
infer_transformation = transforms.Compose([transform.Resize((28,28)),
                                         transforms.ToTensor()])

import os
class PTVisionService(PTServingBaseService):
    def __init__(self, model_name, model_path):
        # Call the constructor of the parent class.
        super(PTVisionService, self).__init__(model_name, model_path)
        # Call the customized function to load the model.
        self.model = Mnist(model_path)
        # Load tags.
        self.label = [0,1,2,3,4,5,6,7,8,9]
        # Labels can also be loaded by label file.
        # Store the label.json file in the model directory. The following information is read:
        dir_path = os.path.dirname(os.path.realpath(self.model_path))
        with open(os.path.join(dir_path, 'label.json')) as f:
            self.label = json.load(f)

    def _preprocess(self, data):
        preprocessed_data = {}  
        for k, v in data.items():
            input_batch = []
            for file_name, file_content in v.items():
                with Image.open(file_content) as image1:
                    # Gray processing
                    image1 = image1.convert("L")
                    if torch.cuda.is_available():
                        input_batch.append(infer_transformation(image1).cuda())
                    else:
                        input_batch.append(infer_transformation(image1))
                input_batch_var = torch.autograd.Variable(torch.stack(input_batch, dim=0), volatile=True)
                print(input_batch_var.shape)
            preprocessed_data[k] = input_batch_var
        return preprocessed_data

    def _postprocess(self, data):
        results = []
        for k, v in data.items():
            result = torch.argmax(v[0])
            result = {k: self.label[result]}
            results.append(result)
        return results

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.hidden1 = nn.Linear(784, 5120, bias=False)
self.output = nn.Linear(5120, 10, bias=False)

def forward(self, x):
    x = x.view(x.size()[0], -1)
    x = F.relu((self.hidden1(x)))
    x = F.dropout(x, 0.2)
    x = self.output(x)
    return F.log_softmax(x)

def Mnist(model_path, **kwargs):
    # Generate a network.
    model = Net()
    # Load the model.
    if torch.cuda.is_available():
        device = torch.device('cuda')
        model.load_state_dict(torch.load(model_path, map_location="cuda:0"))
    else:
        device = torch.device('cpu')
        model.load_state_dict(torch.load(model_path, map_location=device))
    # CPU or GPU mapping
    model.to(device)
    # Declare an inference mode.
    model.eval()
    return model

12.4 Caffe

Training and Saving a Model

**lenet_train_test.prototxt** file

```
name: "LeNet"
layer {
    name: "mnist"
    type: "Data"
    top: "data"
    top: "label"
    include {
        phase: TRAIN
    }
    transform_param {
        scale: 0.00390625
    }
    data_param {
        source: "examples/mnist/mnist_train_lmdb"
        batch_size: 64
        backend: LMDB
    }
}
layer {
    name: "mnist"
    type: "Data"
    top: "data"
    top: "label"
    include {
        phase: TEST
    }
    transform_param {
        scale: 0.00390625
    }
    data_param {
        source: "examples/mnist/mnist_test_lmdb"
        batch_size: 100
    }
```
backend: LMDB
}
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
    name: "ip1"
    type: "InnerProduct"
    bottom: "pool2"
    top: "ip1"
    param {
        lr_mult: 1
    }
    param {
        lr_mult: 2
    }
    inner_product_param {
        num_output: 500
        weight_filler {
            type: "xavier"
        }
        bias_filler {
            type: "constant"
        }
    }
}
layer {
    name: "relu1"
    type: "ReLU"
    bottom: "ip1"
    top: "ip1"
}
layer {
    name: "ip2"
    type: "InnerProduct"
    bottom: "ip1"
    top: "ip2"
    param {
        lr_mult: 1
    }
    param {
        lr_mult: 2
    }
    inner_product_param {
        num_output: 10
        weight_filler {
            type: "xavier"
        }
        bias_filler {
            type: "constant"
        }
    }
}
layer {
    name: "accuracy"
    type: "Accuracy"
    bottom: "ip2"
    bottom: "label"
    top: "accuracy"
    include {
        phase: TEST
    }
}
layer {
    name: "loss"
    type: "SoftmaxWithLoss"
    bottom: "ip2"
    bottom: "label"
    top: "loss"
}

\texttt{lenet_solver.prototxt} file
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 1000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
# solver mode: CPU or GPU
solver_mode: CPU

Train the model.

./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt

The caffemodel file is generated after model training. Rewrite the lenet_train_test.prototxt file to the lenet_deploy.prototxt file used for deployment by modifying input and output layers.
	name: "LeNet"
layer {
  name: "data"
  type: "Input"
top: "data"
input_param { shape: { dim: 1 dim: 1 dim: 28 dim: 28 } }
}
layer {
  name: "conv1"
  type: "Convolution"
bottom: "data"
top: "conv1"
param {
  lr_mult: 1
}
  param {
    lr_mult: 2
  }
convolution_param {
  num_output: 20
  kernel_size: 5
  stride: 1
  weight_filler {
    type: "xavier"
  }
  bias_filler {
    type: "constant"
  }
}
layer {
  name: "pool1"
  type: "Pooling"
bottom: "conv1"
top: "pool1"
pooling_param {  
  pool: MAX  
  kernel_size: 2  
  stride: 2  
}

layer {  
  name: "conv2"  
  type: "Convolution"  
  bottom: "pool1"  
  top: "conv2"  
  param {  
    lr_mult: 1  
  }  
  param {  
    lr_mult: 2  
  }  
  convolution_param {  
    num_output: 50  
    kernel_size: 5  
    stride: 1  
    weight_filler {  
      type: "xavier"  
    }  
    bias_filler {  
      type: "constant"  
    }  
  }  
}

layer {  
  name: "pool2"  
  type: "Pooling"  
  bottom: "conv2"  
  top: "pool2"  
  pooling_param {  
    pool: MAX  
    kernel_size: 2  
    stride: 2  
  }  
}

layer {  
  name: "ip1"  
  type: "InnerProduct"  
  bottom: "pool2"  
  top: "ip1"  
  param {  
    lr_mult: 1  
  }  
  param {  
    lr_mult: 2  
  }  
  inner_product_param {  
    num_output: 500  
    weight_filler {  
      type: "xavier"  
    }  
    bias_filler {  
      type: "constant"  
    }  
  }  
}

layer {  
  name: "relu1"  
  type: "ReLU"  
  bottom: "ip1"  
  top: "ip1"  
}

layer {  
  name: "ip2"  
}
type: "InnerProduct"
bottom: "ip1"
top: "ip2"
param {
    lr_mult: 1
}
param {
    lr_mult: 2
}
inner_product_param {
    num_output: 10
    weight_filler {
        type: "xavier"
    }
    bias_filler {
        type: "constant"
    }
}
layer {
    name: "prob"
    type: "Softmax"
    bottom: "ip2"
    top: "prob"
}

Inference Code

```python
from model_service.caffe_model_service import CaffeBaseService
import numpy as np
import os, json
import caffe
from PIL import Image

class LenetService(CaffeBaseService):
    def __init__(self, model_name, model_path):
        # Call the inference method of the parent class.
        super(LenetService, self).__init__(model_name, model_path)

        # Configure preprocessing information.
        transformer = caffe.io.Transformer({'data': self.net.blobs['data'].data.shape})
        transformer.set_transpose('data', (2, 0, 1))
        transformer.set_raw_scale('data', 255.0)

        # If the batch size is set to 1, inference is supported for only one image.
        self.net.blobs['data'].reshape(1, 1, 28, 28)
        self.transformer = transformer

        # Define the class labels.
        self.label = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

    def _preprocess(self, data):
        for k, v in data.items():
            for file_name, file_content in v.items():
                im = caffe.io.load_image(file_content, color=False)
                # Pre-process the images.
                self.net.blobs['data'].data[...] = self.transformer.preprocess('data', im)

                return
```
def _postprocess(self, data):
    data = data['prob'][0, :]
    predicted = np.argmax(data)
    predicted = {"predicted": str(predicted) }
    return predicted

12.5 XGBoost

Training and Saving a Model

import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split

# Prepare training data and setting parameters
iris = pd.read_csv('/data/iris.csv')
X = iris.drop(['virginica'], axis=1)
y = iris[['virginica']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234565)
params = {
    'booster': 'gbtree',
    'objective': 'multi:softmax',
    'num_class': 3,
    'gamma': 0.1,
    'max_depth': 6,
    'lambda': 2,
    'subsample': 0.7,
    'colsample_bytree': 0.7,
    'min_child_weight': 3,
    'silent': 1,
    'eta': 0.1,
    'seed': 1000,
    'nthread': 4,
}
plst = params.items()
dtrain = xgb.DMatrix(X_train, y_train)
num_rounds = 500
model = xgb.train(plst, dtrain, num_rounds)
model.save_model('/tmp/xgboost.m')

After the model is saved, it must be uploaded to the OBS directory before being published. The config.json and customize_service.py files must be contained during publishing. For details about the definition method, see Model Package Specifications.

Inference Code

# coding:utf-8
import collections
import json
import xgboost as xgb
from model_service.python_model_service import XgSklServingBaseService
class user_Service(XgSklServingBaseService):
    # request data preprocess
    def _preprocess(self, data):
        list_data = []
        json_data = json.loads(data, object_pairs_hook=collections.OrderedDict)
        for element in json_data["data"]["req_data"]:  
            array = []
            for each in element:
                array.append(element[each])
            list_data.append(array)
        return list_data
list_data.append(array)
return list_data

# predict
def _predict(self, data):
xg_model = xgb.Booster(model_file=self.model_path)
pre_data = xgb.DMatrix(data)
pre_result = xg_model.predict(pre_data)
pre_result = pre_result.tolist()
return pre_result

# predict result process
def _postprocess(self, data):
resp_data = []
for element in data:
    resp_data.append({"predictresult": element})
return resp_data

12.6 PySpark

Training and Saving a Model

from pyspark.ml import Pipeline, PipelineModel
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import LogisticRegression

# Prepare training data using tuples.
# Prepare training data from a list of (label, features) tuples.
training = spark.createDataFrame([(
    1.0, Vectors.dense([0.0, 1.1, 0.1])),
    (0.0, Vectors.dense([2.0, 1.0, -1.0])),
    (0.0, Vectors.dense([2.0, 1.3, 1.0])),
    (1.0, Vectors.dense([0.0, 1.2, -0.5]))], ["label", "features"])

# Create a training instance. The logistic regression algorithm is used for training.
# Create a LogisticRegression instance. This instance is an Estimator.
lr = LogisticRegression(maxIter=10, regParam=0.01)

# Train the logistic regression model.
# Learn a LogisticRegression model. This uses the parameters stored in lr.
model = lr.fit(training)

# Save the model to a local directory.
# Save model to local path.
model.save("/tmp/spark_model")

After the model is saved, it must be uploaded to the OBS directory before being published. The config.json configuration and customize_service.py must be contained during publishing. For details about the definition method, see Model Package Specifications.

Inference Code

# coding:utf-8
import collections
import json
import traceback
import model_service.log as log
from model_service.spark_model_service import SparkServingBaseService
from pyspark.ml.classification import LogisticRegression

logger = log.getLogger(__name__)
class user_Service(SparkServingBaseService):
    # Pre-process data.
    def __preprocess(self, data):
        logger.info("Begin to handle data from user data...")
        # Read data.
        req_json = json.loads(data, object_pairs_hook=collections.OrderedDict)
        try:
            # Convert data to the spark dataframe format.
            predict_spdf = self.spark.createDataFrame(pd.DataFrame(req_json["data"]['req_data']))
        except Exception as e:
            logger.error("check your request data does meet the requirements ?")
            logger.error(traceback.format_exc())
            raise Exception("check your request data does meet the requirements ?")
        return predict_spdf

    # Perform model inference.
    def __inference(self, data):
        try:
            # Load a model file.
            predict_model = LogisticRegression.load(self.model_path)
            # Perform data inference.
            prediction_result = predict_model.transform(data)
        except Exception as e:
            logger.error(traceback.format_exc())
            raise Exception("Unable to load model and do dataframe transformation.")
        return prediction_result

    # Post-process data.
    def __postprocess(self, pre_data):
        logger.info("Get new data to respond...")
        predict_str = pre_data.toPandas().to_json(orient='records')
        predict_result = json.loads(predict_str)
        return predict_result

12.7 Scikit Learn

Training and Saving a Model

import json
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.externals import joblib
iris = pd.read_csv('/data/iris.csv')
X = iris.drop(['virginica'], axis=1)
y = iris[['virginica']]
# Create a LogisticRegression instance and train model
logisticRegression = LogisticRegression(C=1000.0, random_state=0)
logisticRegression.fit(X, y)
# Save model to local path
joblib.dump(logisticRegression, '/tmp/sklearn.m')

After the model is saved, it must be uploaded to the OBS directory before being published. The config.json and customize_service.py files must be contained during publishing. For details about the definition method, see Model Package Specifications.

Inference Code

# coding:utf-8
import collections
import json
from sklearn.externals import joblib
from model_service.python_model_service import XgSklServingBaseService
class user_Service(XgSklServingBaseService):
    # request data preprocess
    def _preprocess(self, data):
        list_data = []
        json_data = json.loads(data, object_pairs_hook=collections.OrderedDict)
        for element in json_data["data"]["req_data"]:  
            array = []
            for each in element:
                array.append(element[each])
            list_data.append(array)
        return list_data

    # predict
    def _inference(self, data):
        sk_model = joblib.load(self.model_path)
        pre_result = sk_model.predict(data)
        pre_result = pre_result.tolist()
        return pre_result

    # predict result process
    def _postprocess(self, data):
        resp_data = []
        for element in data:
            resp_data.append({"predictresult": element})
        return resp_data
13 Permissions Management

13.1 Creating a User and Granting Permissions

This section describes how to use IAM to implement fine-grained permissions control for your ModelArts resources. With IAM, you can:

- Create IAM users for employees based on the organizational structure of your enterprise. Each IAM user has their own security credentials, providing access to ModelArts resources.
- Grant only the permissions required for users to perform a task.
- Entrust a HUAWEI CLOUD account or cloud service to perform professional and efficient O&M on your ModelArts resources.

If your HUAWEI CLOUD account does not require individual IAM users, you can skip this section.

This section describes the procedure for granting permissions (see Figure 13-1).

Prerequisites

- You have learnt about the permissions supported by ModelArts and understood how to choose policies or roles according to your requirements. For details, see ModelArts Permissions.
- The permissions to use ModelArts depend on OBS authorization. Therefore, you need to grant OBS system permissions to users. For details, see OBS Permissions.
- For the system policies of other services, see System Permissions.
Process Flow

Figure 13-1 Process for granting ModelArts permissions

1. **Create a user group and assign permissions to it.**
   Create a user group on the IAM console, and assign the ModelArts CommonOperations policy to the group.
   The use of ModelArts depends on OBS permissions. Therefore, assign the Tenant Administrator policy that takes effect for global services to the user group.

2. **Create a user and add it to a user group.**
   Create a user on the IAM console and add the user to the group created in 1.

3. **Log in and verify permissions.**
   Log in to the ModelArts console by using the newly created user, and verify that the user only has read permissions for ModelArts.
   - Choose **Service List** > **ModelArts**. On the ModelArts management console, choose **Dedicated Resource Pools** > **Create**. If the creation fails (assume that the current permission contains only **ModelArts CommonOperations**), the ModelArts CommonOperations policy has already taken effect.
   - Choose any other service in **Service List**. If a message appears indicating that you have insufficient permissions to access the service (assume that the current permission contains only **ModelArts CommonOperations**), the ModelArts CommonOperations policy has already taken effect.
   - Choose **Service List** > **ModelArts**. On the ModelArts management console, choose **Data Management** > **Datasets** > **Create Dataset**. If the corresponding OBS path can be accessed, the Tenant Administrator policy for global services has already taken effect.
13.2 Creating a Custom Policy

Custom policies can be created as a supplement to the system policies of ModelArts. For the actions that can be added for custom policies, see ModelArts API Reference > Permissions Policies and Supported Actions.

You can create custom policies in either of the following ways:

- Visual editor: Select cloud services, actions, resources, and request conditions. This does not require knowledge of policy syntax.
- JSON: Edit JSON policies from scratch or based on an existing policy.

For details, see Creating a Custom Policy. This section describes example custom policies of OBS (a dependent service of ModelArts) and ModelArts.

Precautions

- The permissions to use ModelArts depend on OBS authorization. Therefore, you need to grant OBS system permissions to users.
- A custom policy can contain actions of multiple services that are globally accessible or accessible through region-specific projects.
- To define permissions required to access both global and project-level services, create two custom policies and specify the scope as Global services and Project-level services. Then grant the two policies to the users.
- By default, an IAM user created under the current account has all the operation permissions for ModelArts. To control the permissions of an IAM user, for example, to deny a user the permission to use a function, create a custom policy of ModelArts.

Example Custom Policies of OBS

ModelArts is a project-level service, and OBS is a global service. Therefore, you need to create custom policies for the two services respectively and grant them to users. The permissions to use ModelArts depend on OBS authorization. The following example shows the minimum permissions for OBS, including the permissions for OBS buckets and objects. After being granted the minimum permissions for OBS, users can access OBS from ModelArts without restrictions.

```json
{
  "Version": "1.1",
  "Statement": [
    {
      "Resource": ["*"],
      "Effect": "Allow"
    }
  ]
}
```
Example Custom Policies of ModelArts

By default, an IAM user created under the current account has all the operation permissions for ModelArts. To control the permissions of an IAM user, for example, to deny a user the permission to use a function, set the parameters as follows:

- Example: Denying ExeML project deletion
  A deny policy must be used in conjunction with other policies to take effect. If the permissions assigned to a user contain both Allow and Deny actions, the Deny actions take precedence over the Allow actions.
  The following method can be used if you need to assign permissions of the **ModelArts FullAccess** policy to a user but also forbid the user from deleting ExeML projects. Create a custom policy for denying ExeML project deletion, and assign both policies to the group the user belongs to. Then the user can perform all operations on ModelArts except deleting ExeML projects. The following is an example deny policy:

```json
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "modelarts:exemlProject:delete"
      ]
    }
  ]
}
```

- Example: Denying use of development environment functions
  The following is a policy configuration example for this user:

```json
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "modelarts:notebook:list",
        "modelarts:notebook:create",
        "modelarts:notebook:get",
        "modelarts:notebook:update",
        "modelarts:notebook:delete",
        "modelarts:notebook:action",
        "modelarts:notebook:access"
      ]
    }
  ]
}
```
### 14.1 ModelArts Metrics

#### Description

The cloud service platform provides Cloud Eye to help you better understand the status of your ModelArts real-time services and models. You can use Cloud Eye to automatically monitor your ModelArts real-time services and models in real time and manage alarms and notifications, so that you can keep track of performance metrics of ModelArts and models.

#### Namespace

SYS.ModelArts

#### Monitoring Metrics

<table>
<thead>
<tr>
<th>Metric ID</th>
<th>Metric Name</th>
<th>Meaning</th>
<th>Value Range</th>
<th>Measurement Object &amp; Dimension</th>
<th>Monitoring Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu_usage</td>
<td>CPU Usage</td>
<td>CPU usage of ModelArts Unit: %</td>
<td>≥ 0%</td>
<td>ModelArts objects model_id</td>
<td>1 minute</td>
</tr>
<tr>
<td>Metric ID</td>
<td>Metric Name</td>
<td>Meaning</td>
<td>Value Range</td>
<td>Measurement Object &amp; Dimension</td>
<td>Monitoring Interval</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
<td>----------------------------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>mem_usage</td>
<td>Memory Usage</td>
<td>Memory usage of ModelArts</td>
<td>≥ 0%</td>
<td>Measurement object: ModelArts models</td>
<td>1 minute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unit: %</td>
<td></td>
<td>Dimension: model_id</td>
<td></td>
</tr>
<tr>
<td>gpu_util</td>
<td>GPU Usage</td>
<td>GPU usage of ModelArts</td>
<td>≥ 0%</td>
<td>Measurement object: ModelArts models</td>
<td>1 minute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unit: %</td>
<td></td>
<td>Dimension: model_id</td>
<td></td>
</tr>
<tr>
<td>gpu_mem_usage</td>
<td>GPU Memory Usage</td>
<td>GPU memory usage of ModelArts</td>
<td>≥ 0%</td>
<td>Measurement object: ModelArts models</td>
<td>1 minute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unit: %</td>
<td></td>
<td>Dimension: model_id</td>
<td></td>
</tr>
<tr>
<td>successfully_called_times</td>
<td>Number of Successful Calls</td>
<td>Times that ModelArts has been successfully called</td>
<td>≥ Count/min</td>
<td>Measurement object: ModelArts models, ModelArts real-time services, Dimension: model_id, service_id</td>
<td>1 minute</td>
</tr>
<tr>
<td>failed_called_times</td>
<td>Number of Failed Calls</td>
<td>Times that ModelArts failed to be called</td>
<td>≥ Count/min</td>
<td>Measurement object: ModelArts models, ModelArts real-time services, Dimension: model_id, service_id</td>
<td>1 minute</td>
</tr>
</tbody>
</table>
### Dimensions

**Table 14-2 Dimension description**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_id</td>
<td>Real-time service ID</td>
</tr>
<tr>
<td>model_id</td>
<td>Model ID</td>
</tr>
</tbody>
</table>
14.2 Setting Alarm Rules

Scenario

Setting alarm rules allows you to customize the monitored objects and notification policies so that you can know the status of ModelArts real-time services and models in a timely manner.

An alarm rule includes the alarm rule name, monitored object, metric, threshold, monitoring interval, and whether to send a notification. This section describes how to set alarm rules for ModelArts services and models.

Prerequisites

You have created an ModelArts real-time service.

Procedure

1. Log in to the management console.
2. Click Service List. Under Management & Deployment, click Cloud Eye.
3. In the left navigation pane, choose Cloud Service Monitoring > ModelArts.
4. Select a real-time service for which you want to create an alarm rule and click Create Alarm Rule in the Operation column.
5. On the Create Alarm Rule page, create an alarm rule for ModelArts real-time services and models as prompted.

Figure 14-1 Creating an alarm rule

6. After the setting is complete, click Create. When an alarm that meets the rule is generated, the system automatically sends a notification.
14.3 Viewing Monitoring Metrics

Scenario

Cloud Eye on the cloud service platform monitors the status of ModelArts real-time services and model loads. You can obtain the monitoring metrics of each ModelArts real-time service and model loads on the management console. Monitored data requires a period of time for transmission and display. The status of ModelArts displayed on the Cloud Eye console is usually the status obtained 5 to 10 minutes before. You can view the monitored data of a newly created real-time service 5 to 10 minutes later.

Prerequisites

- The ModelArts real-time service is running properly.
- Alarm rules have been configured on the Cloud Eye page. For details, see Setting Alarm Rules.
- The real-time service has been properly running for at least 10 minutes.
- The monitoring data and graphics are available for a new real-time service after the service runs for at least 10 minutes.
- Cloud Eye does not display the metrics of a faulty or deleted real-time service. The monitoring metrics can be viewed after the real-time service starts or recovers.

Monitoring data is unavailable without alarm rules configured on Cloud Eye. For details, see Setting Alarm Rules.

Procedure

1. Log in to the management console.
2. Click Service List. Under Management & Deployment, click Cloud Eye.
3. In the left navigation pane, choose Cloud Service Monitoring > ModelArts.
4. View monitoring graphs.
   - Viewing monitoring graphs of the real-time service: Click View Graph in the Operation column.
   - Viewing monitoring graphs of the model loads: Click 🔄 next to the target real-time service, and select View Graph from the drop-down list for model loads in the Operation column.
5. In the monitoring area, you can select a duration to view the monitoring data. You can view the monitoring data in the recent 1 hour, 3 hours, or 12 hours. To view the monitoring curve of a longer time range, click 🕒 to enlarge the graph.
15 Audit Logs

15.1 Key Operations Recorded by CTS

With CTS, you can record operations associated with ModelArts for later query, audit, and backtrack operations.

Prerequisites

CTS has been enabled.

Key Operations Recorded for Data Management

Table 15-1  Key operations that can be audited for data management

<table>
<thead>
<tr>
<th>Operation</th>
<th>Resource Type</th>
<th>Trace Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating a dataset</td>
<td>dataset</td>
<td>createDataset</td>
</tr>
<tr>
<td>Deleting a dataset</td>
<td>dataset</td>
<td>deleteDataset</td>
</tr>
<tr>
<td>Updating a dataset</td>
<td>dataset</td>
<td>updateDataset</td>
</tr>
<tr>
<td>Publishing a version of a dataset</td>
<td>dataset</td>
<td>publishDatasetVersion</td>
</tr>
<tr>
<td>Deleting a dataset version</td>
<td>dataset</td>
<td>deleteDatasetVersion</td>
</tr>
<tr>
<td>Synchronizing the data source</td>
<td>dataset</td>
<td>syncDataSource</td>
</tr>
<tr>
<td>Exporting a dataset</td>
<td>dataset</td>
<td>exportDataFromDataset</td>
</tr>
<tr>
<td>Creating an auto labeling task</td>
<td>dataset</td>
<td>createAutoLabelingTask</td>
</tr>
<tr>
<td>Creating an auto grouping task</td>
<td>dataset</td>
<td>createAutoGroupingTask</td>
</tr>
<tr>
<td>Operation</td>
<td>Resource Type</td>
<td>Trace Name</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Creating an automatic deployment task</td>
<td>dataset</td>
<td>createAutoDeployTask</td>
</tr>
<tr>
<td>Importing samples to a dataset</td>
<td>dataset</td>
<td>importSamplesToDataset</td>
</tr>
<tr>
<td>Creating a dataset label</td>
<td>dataset</td>
<td>createLabel</td>
</tr>
<tr>
<td>Updating a dataset label</td>
<td>dataset</td>
<td>updateLabel</td>
</tr>
<tr>
<td>Deleting a dataset label</td>
<td>dataset</td>
<td>deleteLabel</td>
</tr>
<tr>
<td>Deleting a dataset label and its samples</td>
<td>dataset</td>
<td>deleteLabelWithSamples</td>
</tr>
<tr>
<td>Adding samples</td>
<td>dataset</td>
<td>uploadSamples</td>
</tr>
<tr>
<td>Deleting samples</td>
<td>dataset</td>
<td>deleteSamples</td>
</tr>
<tr>
<td>Stopping an auto labeling task</td>
<td>dataset</td>
<td>stopTask</td>
</tr>
<tr>
<td>Creating a team labeling task</td>
<td>dataset</td>
<td>createWorkforceTask</td>
</tr>
<tr>
<td>Deleting a team labeling task</td>
<td>dataset</td>
<td>deleteWorkforceTask</td>
</tr>
<tr>
<td>Starting the acceptance of team labeling</td>
<td>dataset</td>
<td>startWorkforceSamplingTask</td>
</tr>
<tr>
<td>Approving/rejecting/canceling acceptance</td>
<td>dataset</td>
<td>updateWorkforceSamplingTask</td>
</tr>
<tr>
<td>Submitting sample review comments for acceptance</td>
<td>dataset</td>
<td>acceptSamples</td>
</tr>
<tr>
<td>Adding a label to a sample</td>
<td>dataset</td>
<td>updateSamples</td>
</tr>
<tr>
<td>Sending an email to labeling team members</td>
<td>dataset</td>
<td>sendEmails</td>
</tr>
<tr>
<td>Starting the team labeling task as the contact person</td>
<td>dataset</td>
<td>startWorkforceTask</td>
</tr>
<tr>
<td>Updating a team labeling task</td>
<td>dataset</td>
<td>updateWorkforceTask</td>
</tr>
<tr>
<td>Adding a label to a team-labeled sample</td>
<td>dataset</td>
<td>updateWorkforceTask-Samples</td>
</tr>
<tr>
<td>Reviewing team labeling results</td>
<td>dataset</td>
<td>reviewSamples</td>
</tr>
<tr>
<td>Operation</td>
<td>Resource Type</td>
<td>Trace Name</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>---------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Creating a labeling team member</td>
<td>workforce</td>
<td>createWorker</td>
</tr>
<tr>
<td>Updating a labeling team member</td>
<td>workforce</td>
<td>updateWorker</td>
</tr>
<tr>
<td>Deleting a labeling team member</td>
<td>workforce</td>
<td>deleteWorker</td>
</tr>
<tr>
<td>Batch deleting labeling team members</td>
<td>workforce</td>
<td>batchDeleteWorker</td>
</tr>
<tr>
<td>Creating a labeling team</td>
<td>workforce</td>
<td>createWorkforce</td>
</tr>
<tr>
<td>Updating a labeling team</td>
<td>workforce</td>
<td>updateWorkforce</td>
</tr>
<tr>
<td>Deleting a labeling team</td>
<td>workforce</td>
<td>deleteWorkforce</td>
</tr>
<tr>
<td>Automatically creating an IAM agency</td>
<td>IAM</td>
<td>createAgency</td>
</tr>
<tr>
<td>Logging in to the labelConsole labeling platform as a labeling team member</td>
<td>labelConsoleWorker</td>
<td>workerLoginLabelConsole</td>
</tr>
<tr>
<td>Logging out of the labelConsole labeling platform as a labeling team member</td>
<td>labelConsoleWorker</td>
<td>workerLogOutLabelConsole</td>
</tr>
<tr>
<td>Changing the password of the labelConsole platform as a labeling team member</td>
<td>labelConsoleWorker</td>
<td>workerChangePassword</td>
</tr>
<tr>
<td>Forgetting the password of the labelConsole platform as a labeling team member</td>
<td>labelConsoleWorker</td>
<td>workerForgetPassword</td>
</tr>
<tr>
<td>Resetting the password of the labelConsole platform through the URL as a labeling team member</td>
<td>labelConsoleWorker</td>
<td>workerResetPassword</td>
</tr>
</tbody>
</table>
Key Operations Recorded for Development Environments

Table 15-2 Key operations that can be audited in the development environment

<table>
<thead>
<tr>
<th>Operation</th>
<th>Resource Type</th>
<th>Trace Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating a notebook instance</td>
<td>Notebook</td>
<td>create_instance</td>
</tr>
<tr>
<td>Deleting a notebook instance</td>
<td>Notebook</td>
<td>delete_instance</td>
</tr>
<tr>
<td>Starting a notebook instance</td>
<td>Notebook</td>
<td>change_instance_status</td>
</tr>
<tr>
<td>Stopping a notebook instance</td>
<td>Notebook</td>
<td>change_instance_status</td>
</tr>
</tbody>
</table>

Key Operations Recorded for Training Jobs

Table 15-3 Key operations that can be audited in a training job

<table>
<thead>
<tr>
<th>Operation</th>
<th>Resource Type</th>
<th>Trace Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating a training job</td>
<td>ModelArtsTrainJob</td>
<td>createModelArtsTrainJob</td>
</tr>
<tr>
<td>Creating a version of a training job</td>
<td>ModelArtsTrainJob</td>
<td>createModelArtsTrainVersion</td>
</tr>
<tr>
<td>Stopping a training job</td>
<td>ModelArtsTrainJob</td>
<td>stopModelArtsTrainVersion</td>
</tr>
<tr>
<td>Modifying the description of a training job</td>
<td>ModelArtsTrainJob</td>
<td>updateModelArtsTrainDesc</td>
</tr>
<tr>
<td>Deleting a training job version</td>
<td>ModelArtsTrainJob</td>
<td>deleteModelArtsTrainVersion</td>
</tr>
<tr>
<td>Deleting a training job</td>
<td>ModelArtsTrainJob</td>
<td>deleteModelArtsTrainJob</td>
</tr>
<tr>
<td>Creating a training job configuration</td>
<td>ModelArtsTrainConfig</td>
<td>createModelArtsTrainConfig</td>
</tr>
<tr>
<td>Modifying a training job configuration</td>
<td>ModelArtsTrainConfig</td>
<td>updateModelArtsTrainConfig</td>
</tr>
<tr>
<td>Deleting a training job configuration</td>
<td>ModelArtsTrainConfig</td>
<td>deleteModelArtsTrainConfig</td>
</tr>
<tr>
<td>Creating a visualization job</td>
<td>ModelArtsTensorboardJob</td>
<td>createModelArtsTensorboardJob</td>
</tr>
</tbody>
</table>
### Key Operations Recorded for Model Management

**Table 15-4** Key operations that can be audited for model management

<table>
<thead>
<tr>
<th>Operation</th>
<th>Resource Type</th>
<th>Trace Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importing a model</td>
<td>model</td>
<td>addModel</td>
</tr>
<tr>
<td>Updating a model</td>
<td>model</td>
<td>updateModel</td>
</tr>
<tr>
<td>Deleting a model</td>
<td>model</td>
<td>deleteModel</td>
</tr>
<tr>
<td>Adding a model conversion task</td>
<td>convert</td>
<td>addConvert</td>
</tr>
<tr>
<td>Updating a model conversion task</td>
<td>convert</td>
<td>updateConvert</td>
</tr>
<tr>
<td>Deleting a model conversion task</td>
<td>convert</td>
<td>deleteConvert</td>
</tr>
</tbody>
</table>

### Key Operations Recorded for Service Management

**Table 15-5** Key operations that can be audited for service management

<table>
<thead>
<tr>
<th>Operation</th>
<th>Resource Type</th>
<th>Trace Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploying a model as a service</td>
<td>service</td>
<td>addService</td>
</tr>
<tr>
<td>Deleting a service</td>
<td>service</td>
<td>deleteService</td>
</tr>
<tr>
<td>Updating a service</td>
<td>service</td>
<td>updateService</td>
</tr>
<tr>
<td>Starting/stopping a service</td>
<td>service</td>
<td>startOrStopService</td>
</tr>
</tbody>
</table>
### 15.2 Viewing Audit Logs

After CTS is enabled, CTS starts recording operations related to ModelArts. The CTS management console stores the last seven days of operation records. This section describes how to query operation records of the last seven days on the CTS management console.

#### Procedure

1. Log in to the CTS management console.
2. Click in the upper left corner of the console and select a region.
3. In the left navigation pane, click **Trace List**.
4. Specify the filter criteria used for querying traces. The following four filter criteria are available:
   - **Trace Source**, **Resource Type**, and **Search By**
     Select a filter criterion from the drop-down list.
     - If you select **Trace name** for **Search By**, you need to select a specific trace name.
     - If you select **Resource ID** for **Search By**, you need to enter a specific resource ID.
     - If you select **Resource name** for **Search By**, you need to select or enter a specific resource name.
   - **Operator**: Select a specific operator (a user rather than an account).
   - **Trace Status**: Available options include all **trace statuses**, **normal**, **warning**, and **incident**. You can only select one of them.
   - **Time Range**: You can view traces generated during any time range of the last seven days.
5. Click on the left of a trace to expand its details.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Resource Type</th>
<th>Trace Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding an access key</td>
<td>service</td>
<td>addAkSk</td>
</tr>
<tr>
<td>Deleting an access key</td>
<td>service</td>
<td>deleteAkSk</td>
</tr>
<tr>
<td>Creating a dedicated resource pool</td>
<td>cluster</td>
<td>createCluster</td>
</tr>
<tr>
<td>Deleting a dedicated resource pool</td>
<td>cluster</td>
<td>deleteCluster</td>
</tr>
<tr>
<td>Adding a node to a dedicated resource pool</td>
<td>cluster</td>
<td>addClusterNode</td>
</tr>
<tr>
<td>Deleting a node from a dedicated resource pool</td>
<td>cluster</td>
<td>deleteClusterNode</td>
</tr>
</tbody>
</table>
6. Click **View Trace** in the **Operation** column. In the displayed **View Trace** dialog box, the trace structure details are displayed.

For details about the key fields in the CTS trace structure, see the **Cloud Trace Service User Guide**.
### Change History

<table>
<thead>
<tr>
<th>Release Date</th>
<th>What's New</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-11-10</td>
<td>- Optimized the display of the development environment when a notebook is created.</td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Creating a Notebook Instance</a></td>
</tr>
<tr>
<td>2020-09-21</td>
<td>- Added the application authentication function for real-time services.</td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Accessing a Real-Time Service (Application Authentication)</a></td>
</tr>
<tr>
<td>2020-09-11</td>
<td>- Added the model evaluation and diagnosis function.</td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Model Evaluation Overview</a></td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Creating a Model Evaluation Job</a></td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Viewing Evaluation Results</a></td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Evaluation Metrics</a></td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Managing Evaluation Job Versions</a></td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Sample Code for Model Evaluation</a></td>
</tr>
<tr>
<td></td>
<td>- Added the function of viewing the evaluation result when viewing a training job.</td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Viewing the Evaluation Result</a></td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Adding the Evaluation Code</a></td>
</tr>
<tr>
<td>2020-08-06</td>
<td>- Updated the description of the notebook function, optimized the document structure, and added common JupyterLab operations.</td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Introduction to Jupyter Notebook</a></td>
</tr>
<tr>
<td></td>
<td>- <a href="#">JupyterLab Overview and Common Operations</a></td>
</tr>
<tr>
<td></td>
<td>- Added the dataset of the table type, which supports the DWS data source input.</td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Table</a></td>
</tr>
<tr>
<td></td>
<td>- Added guidance on how to integrate real-time service APIs.</td>
</tr>
<tr>
<td></td>
<td>- <a href="#">Integrating a Real-Time Service</a></td>
</tr>
<tr>
<td>Release Date</td>
<td>What's New</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
</tr>
</tbody>
</table>
| 202-07-06    | Added support for video data on the Dataset Management page.  
  - Creating a Dataset  
  - Video Labeling  
  Optimized auto search jobs and added one sample.  
  - Example: Using Backbone Replacement to Optimize Built-in Algorithms on Ascend 910 Without Modifying Code  
  Added the support for the GitHub code library in the development environment.  
  - Creating a Notebook Instance with a Git Repository  
  - Using the Git Plug-in  
  Added the support for the PyTorch 1.4.0 engine in development environments and model import.  
  - Creating a Notebook Instance  
  - Importing a Meta Model from OBS  
  Added the sample code of the TensorFlow 2.X custom script.  
  - TensorFlow 2.1 |
<table>
<thead>
<tr>
<th>Release Date</th>
<th>What's New</th>
</tr>
</thead>
</table>
| 2020-06-08   | Added table dataset and CSV file import and release.  
  - **Creating a Dataset**  
  - **Specifications for Importing Data from an OBS Directory**  
  Added task startup and management to data feature analysis.  
  - **Data Features**  
  Enhanced **DevEnviron** by introducing JupyterLab and TensorFlow 2.1, integrated multiple basic capabilities of ModelArts, and improved user experience.  
  - **Creating a Notebook Instance**  
  - **Synchronizing Files with OBS**  
  - **JupyterLab Overview and Common Operations**  
  - **Uploading Large Files to a Notebook Instance**  
  - **Downloading Large Files in Notebook Instances to a Local PC**  
  Optimized auto search job: Added hyperparameter search with **fix_norm**, data augmentation with **adv_aug**, and multiple NAS methods such as BETANAS to achieve the optimal precision in mobile setting on ImageNet. Multisearch is available using at least five lines of code.  
  - **Introduction to Auto Search Jobs**  
  - **Creating an Auto Search Job**  
  - **Code Compilation Specifications**  
  - **YAML Configuration File Description**  
  - **Example: Replacing the Original ResNet-50 with a Better Network Architecture**  
  - **Example: Searching for Hyperparameters Using Classic Hyperparameter Algorithms**  
  - **Example: Searching for Network Architectures Using the MBNAS Algorithm**  
  - **Example: Implementing Auto Data Augmentation Using a Preset Data Augmentation Policy**  
  - **Example: Using Multisearch**  
  Added TensorFlow 2.1. When creating a notebook instance or using a frequently-used framework to create a training job, TensorFlow 2.1 is ready to use.  
  - **Using Frequently-used Frameworks to Train Models**  
  - **Introduction to Notebook**  
  Added the Python 3.7 runtime environment to **Model Management**. |
<table>
<thead>
<tr>
<th>Release Date</th>
<th>What's New</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-03-24</td>
<td>- Importing a Meta Model from OBS</td>
</tr>
<tr>
<td></td>
<td>- Optimized the description of training jobs based on different algorithm sources.</td>
</tr>
<tr>
<td></td>
<td>- Using Built-in Algorithms to Train Models</td>
</tr>
<tr>
<td></td>
<td>- Using Frequently-used Frameworks to Train Models</td>
</tr>
<tr>
<td></td>
<td>- Using Custom Images to Train Models</td>
</tr>
<tr>
<td></td>
<td>- Added support for team labeling for datasets of the text triplet, text classification, and named entity types, and updated the description of the data labeling function.</td>
</tr>
<tr>
<td></td>
<td>Data Management Overview</td>
</tr>
<tr>
<td></td>
<td>- Updated all screenshots in this document because ModelArts UI is upgraded.</td>
</tr>
<tr>
<td>2020-01-07</td>
<td>- Hid the old data management module.</td>
</tr>
<tr>
<td>2019-12-03</td>
<td>- Added four sections for the model import function based on scenarios.</td>
</tr>
<tr>
<td></td>
<td>Importing a Meta Model from a Training Job</td>
</tr>
<tr>
<td></td>
<td>Importing a Meta Model from a Template</td>
</tr>
<tr>
<td></td>
<td>Importing a Meta Model from a Container Image</td>
</tr>
<tr>
<td></td>
<td>Importing a Meta Model from OBS</td>
</tr>
<tr>
<td></td>
<td>- Supported team labeling task management for datasets that support team labeling.</td>
</tr>
<tr>
<td></td>
<td>Managing Team Labeling Tasks</td>
</tr>
<tr>
<td></td>
<td>- Added the text triplet dataset type.</td>
</tr>
<tr>
<td></td>
<td>Text Triplet</td>
</tr>
<tr>
<td></td>
<td>- Optimized the sections about data management.</td>
</tr>
<tr>
<td></td>
<td>Data Management Overview - Deleting a Dataset</td>
</tr>
<tr>
<td></td>
<td>- Added the function of managing auto search jobs.</td>
</tr>
<tr>
<td></td>
<td>Introduction to Auto Search Jobs - Example: Replacing the Original ResNet-50 with a Better Network Architecture</td>
</tr>
<tr>
<td>Release Date</td>
<td>What’s New</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
</tr>
</tbody>
</table>
| 2019-10-17   | • Optimized and added functions for data management based on software changes. Updated descriptions in all related sections and added the following content:  
  Deploying a Model in One Click  
  Data Features  
  Team Labeling Overview  
  • Added sample code of custom scripts (including frequently-used engines).  
  Examples of Custom Scripts  
  • Added monitoring description.  
  ModelArts Metrics |
| 2019-09-30   | • Enriched and optimized the description of model templates.  
  Model Template Overview  
  • Optimized the description of model package specifications and provided more model package examples.  
  Model Package Specifications |
| 2019-08-20   | Added data management and model templates.  
  • Data Management  
  • Model Templates |
| 2019-06-13   | This is the first official release. |