Huawei AI Certification Training

# HCIP-AI-EI Developer

# Image Processing Lab Guide

ISSUE:2.0

HUAWEI TECHNOLOGIES CO., LTD.

**Trademarks and Permissions**

HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

**Notice**

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

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

**Huawei Technologies Co., Ltd.**

Address: Huawei Industrial Base Bantian, Longgang Shenzhen 518129
People's Republic of China

Website: http://e.huawei.com

# Huawei Certificate System

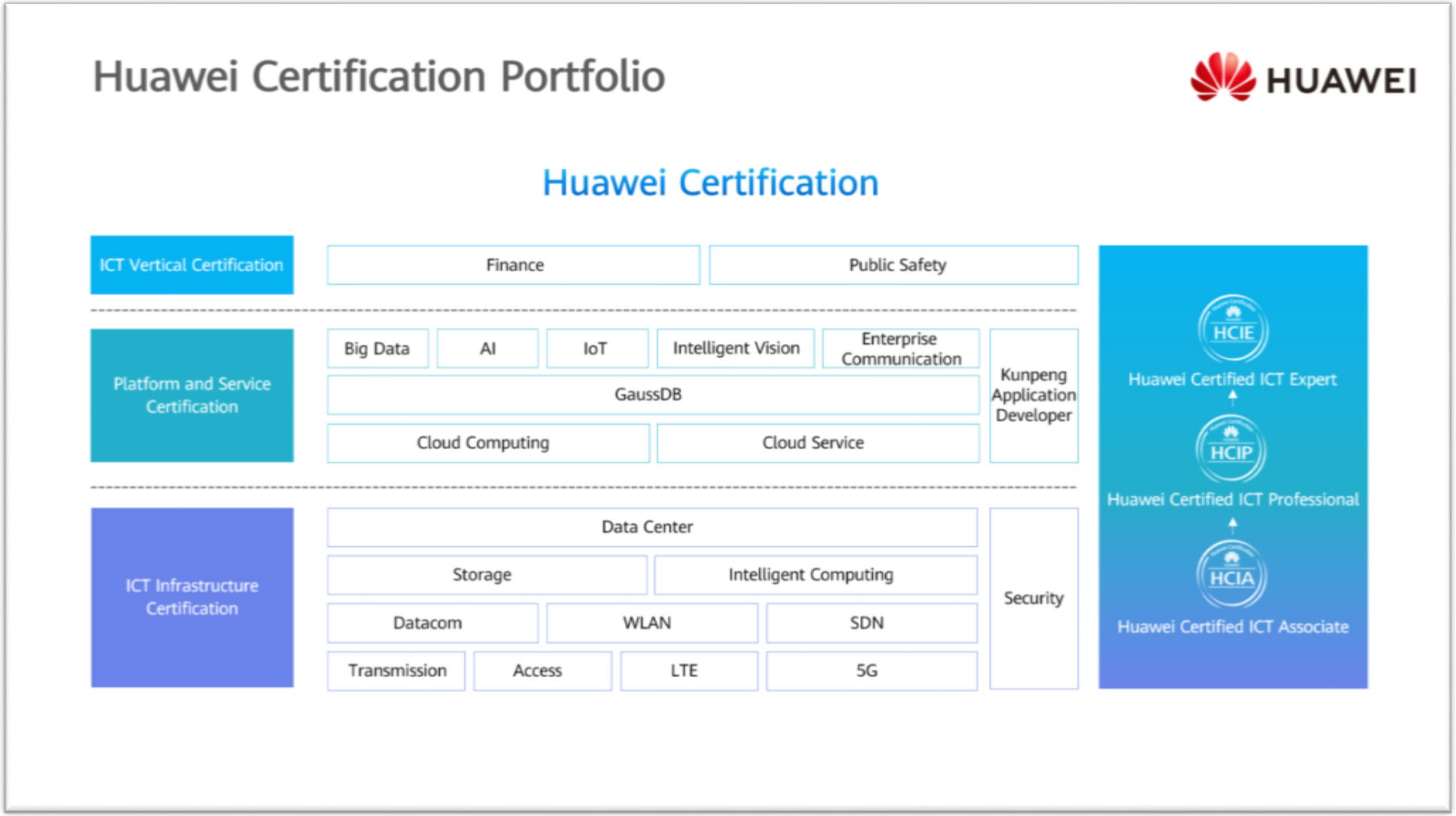
Huawei's certification system is the industry's only one that covers all ICT technical fields. It is developed relying on Huawei's 'platform + ecosystem' strategy and new ICT technical architecture featuring cloud-pipe-device synergy. It provides three types of certifications: ICT Infrastructure Certification, Platform and Service Certification, and ICT Vertical Certification.

To meet ICT professionals' progressive requirements, Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

HCIP-AI-EI Developer V2.0 certification is intended to cultivate professionals who have acquired basic theoretical knowledge about image processing, speech processing, and natural language processing and who are able to conduct development and innovation using Huawei enterprise AI solutions (such as HUAWEI CLOUD EI), general open-source frameworks, and ModelArts, a one-stop development platform for AI developers.

The content of HCIP-AI-EI Developer V2.0 certification includes but is not limited to: neural network basics, image processing theory and applications, speech processing theory and applications, natural language processing theory and applications, ModelArts overview, and image processing, speech processing, natural language processing, and ModelArts platform development experiments. ModelArts is a one-stop development platform for AI developers. With data preprocessing, semi-automatic data labeling, large-scale distributed training, automatic modeling, and on-demand model deployment on devices, edges, and clouds, ModelArts helps AI developers build models quickly and manage the lifecycle of AI development. Compared with V1.0, HCIP-AI-EI Developer V2.0 adds the ModelArts overview and development experiments. In addition, some new EI cloud services are updated.

HCIP-AI-EI Developer V2.0 certification proves that you have systematically understood and mastered neural network basics, image processing theory and applications, speech processing theory and applications, ModelArts overview, natural language processing theory and applications, image processing application development, speech processing application development, natural language processing application development, and ModelArts platform development. With this certification, you will acquire (1) the knowledge and skills for AI pre-sales technical support, AI after-sales technical support, AI product sales, and AI project management; (2) the ability to serve as an image processing developer, speech processing developer, or natural language processing developer.



## Huawei Certification Portfolio

### Huawei Certification

# About This Document

## Overview

This document is a training course for HCIP-AI certification. It is prepared for trainees who are going to take the HCIP-AI exam or readers who want to understand basic AI knowledge. By mastering the content of this manual, you will be able to preprocess images and develop image tagging, text recognition, and image content moderation using HUAWEI CLOUD SERVICES. In the experiment of image preprocessing, we mainly use OpenCV library, while in the lab of image tagging, you can submit RESTful requests to invoke related services of HUAWEI CLOUD. Huawei Enterprise Cloud EI provides various APIs for image processing applications.

## Description

This lab guide consists of three experiments, including image preprocessing lab based on OpenCV library, Smart Album based on HUAWEI CLOUD EI image tag tasks services. These labs aim to improve the practical capability processing image when using AI.

- Experiment 1: Image data preprocessing.
- Experiment 2: Using HUAWEI CLOUD EI image tagging services to implement smart albums.

## Background Knowledge Required

This course is a Huawei certification development course. To better master the contents of this course, readers of this course must meet the following requirements:

- Basic programming capability
- Be familiar of data structure

## Experiment Environment Overview

- Python3.6, OpenCV, numpy, matplotlib, pillow
- HUAWEI CLOUD modelarts (recommended)

# Contents

# 1 Image Data Preprocessing

## 1.1 Introduction

The main purpose of image preprocessing is to eliminate irrelevant information in images, restore useful information, enhance information detectability, and simplify data to the maximum extent, thus improving the reliability of feature extraction and image segmentation, matching, and recognition.

In this experiment, the OpenCV image processing library is used to implement basic image preprocessing operations, including color space conversion, coordinate transformation, grayscale transformation, histogram transformation, and image filtering.

## 1.2 Objective

In this experiment, the image preprocessing technology introduced in the theoretical textbook is implemented by the OpenCV image processing library of Python. This exercise will help you learn how to use OpenCV to preprocess images. This experiment helps trainees understand and master the methods and skills of using Python to develop image preprocessing technologies.

## 1.3 Lab Environment Description

In this experiment, you are advised to install the Python environment of a version later than 3.6 and install external libraries OpenCV, numpy, and maplotlib.

## 1.4 Procedure

### 1.4.1 Basic Operations

**Note: All images read in the code in Lab 1.4 can be read from the local images of the trainees.**

Step 1 Define the matshow function to facilitate picture display.

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
%matplotlib inline
```

```
# use matplotlib to show opencv pic
def matshow(title='image',image=None,gray=False):

    if isinstance(image,np.ndarray):
        if len(image.shape) ==2:
            pass
        elif gray == True:
            # transfer color space to gray in opencv
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        else:
            # transfer color space to RGB in opencv
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    plt.figure()
    plt.imshow(image,cmap='gray')

    plt.axis('off') # close axis 不显示坐标轴
    plt.title(title) # title
    plt.show()
```

Step 2 Image reading and display

```
import cv2
# read one image
# the secend parameter show the way to read, 1 means read as a color image, 0 means gray
im = cv2.imread(r"lena.png",1)
matshow("test",im)
```

Output:

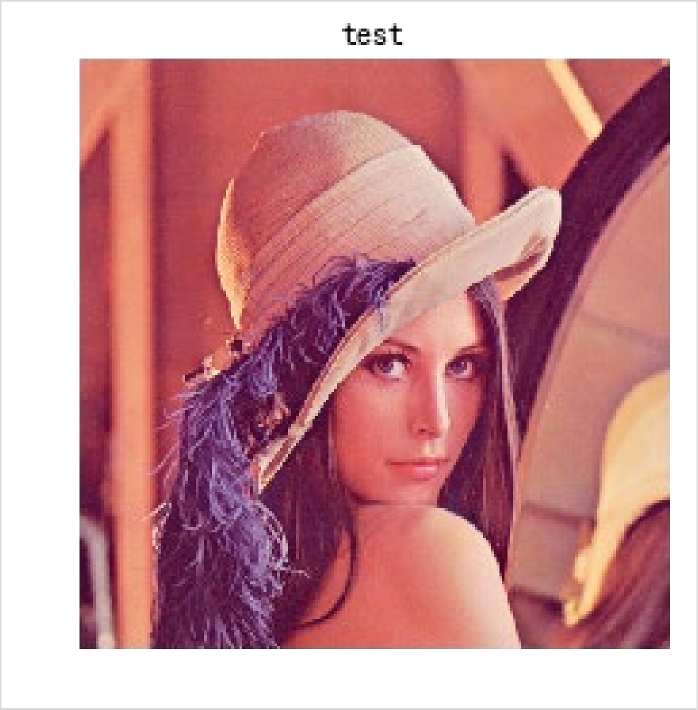


**Figure 1-1 Lena Image 1**

Step 3 Display data types and image sizes

```
# Print the data structure type of the image data.
print(type(im))
# Size of the printed image.
print(im.shape)
```

Output:

```
<class'numpy.ndarray'>
(512, 512, 3)
```

Step 4 Image storage

```
# Save the image to the specified path.
cv2.imwrite('lena.jpg', im)
```

Output:

```
True
```

## 1.4.2 color space conversion

Step 1 color image graying

```
import cv2
im = cv2.imread(r"lena.jpg")
matshow("BGR", im)
# Use cvtColor to change the color space. cv2. COLOR_BGR2GRAY indicates BGR to gray.
img_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
matshow("Gray", img_gray)
```

Output:

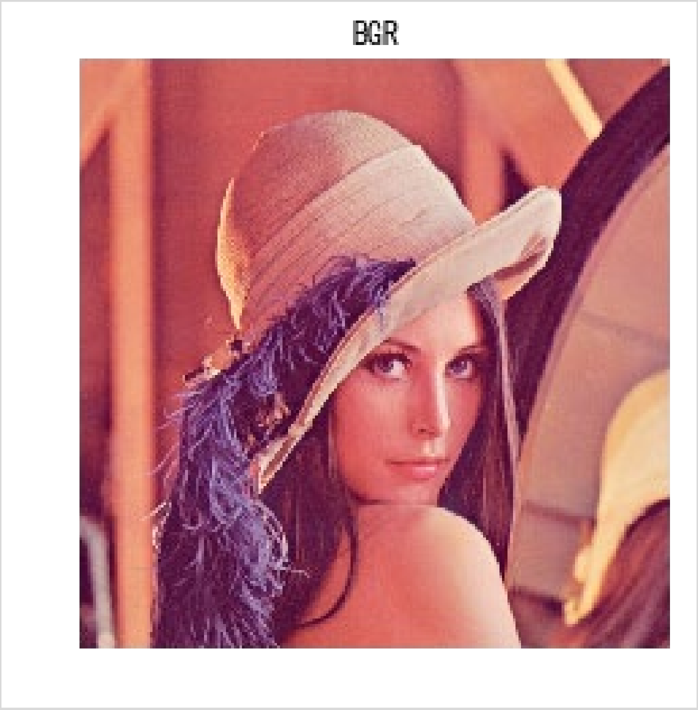


**Figure 1-2 Original lena image**

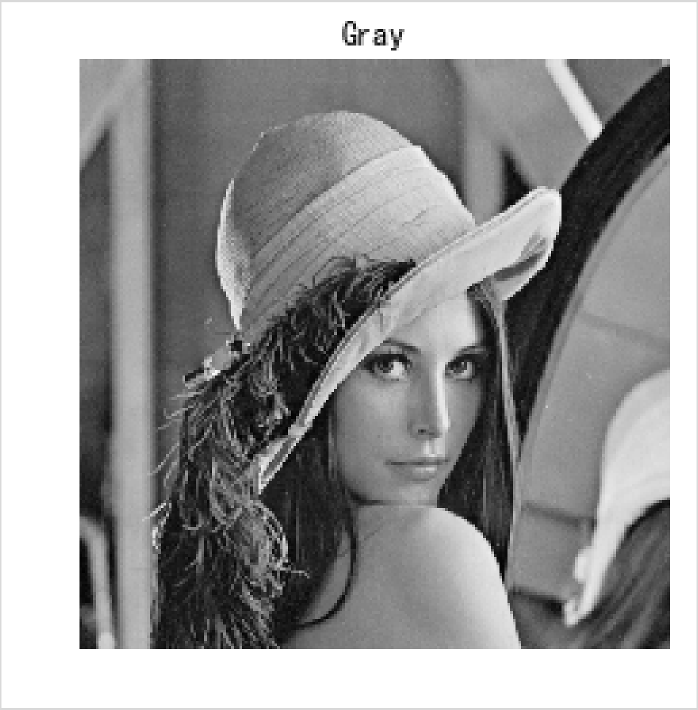


**Figure 1-3 Gray scale lena image**

Step 2 Replace the three-channel sequential BGR with the RGB.

```
import cv2
im = cv2.imread(r"lena.jpg")
matshow("BGR", im)
# Use cvtColor to change the color space. cv2. COLOR_BGR2RGB indicates BGR to RGB.
```

```
im_rgb = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
# When the image data is in three channels, the imshow function considers that the data is BGR.
# Run the imshow command to display RGB data. It is found that the image color is distorted.
matshow("RGB", im_rgb)
```

Output:

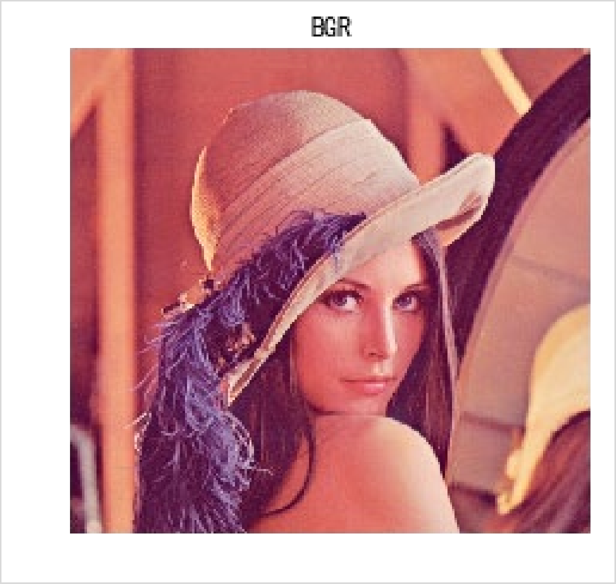


**Figure 1-4 Original lena image**

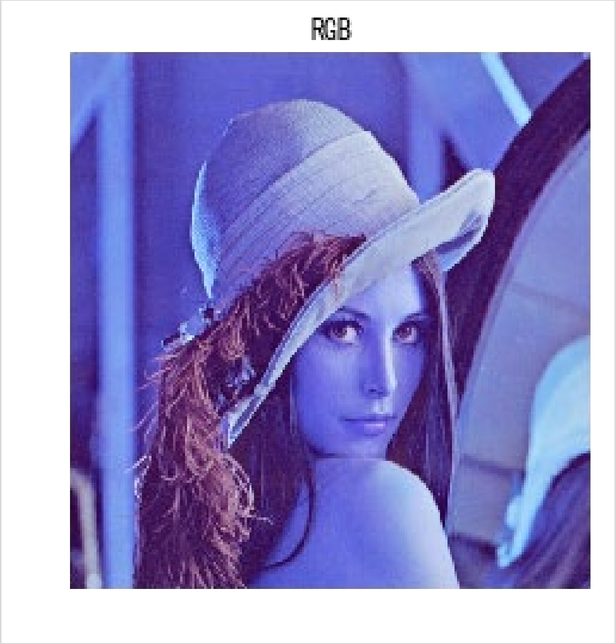


**Figure 1-5 Displaying the RGB lena image using the BGR channel**

Step 3 BGR and HSV color space conversion

```
import cv2
```

```
im = cv2.imread(r"lena.jpg")
matshow("BGR", im)
# Use cvtColor to change the color space. cv2. COLOR_BGR2HSV indicates BGR to HSV.
im_hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
# When the image data is in three channels, the imshow function considers that the data is BGR.
# Run the imshow command to display HSV data. The HSV component is forcibly displayed as the
BGR.
matshow("HSV", im_hsv)
```

Output:

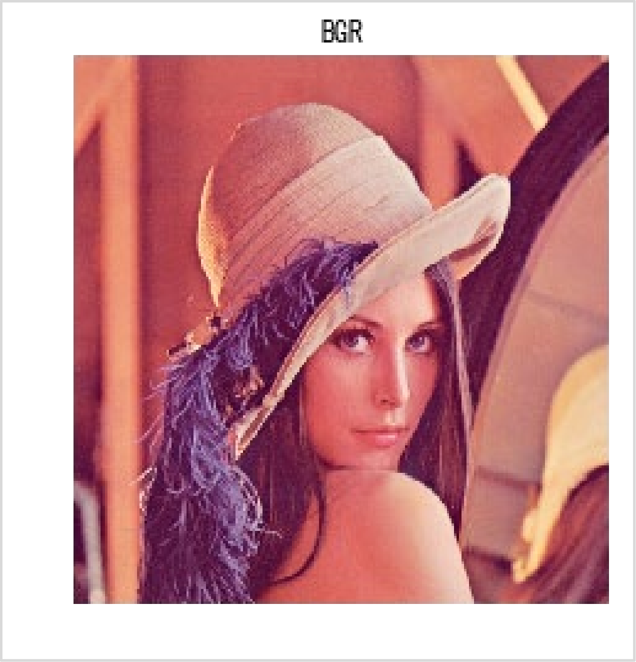


**Figure 1-6 Original lena image**

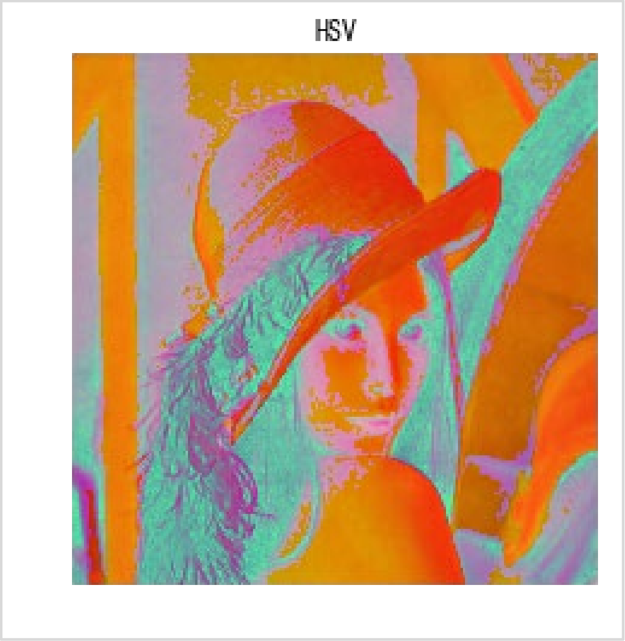


**Figure 1-7 Displaying the HSV lena image using the BGR channel**

## 1.4.3 coordinate Transformation

Step 1 translation

```
import numpy as np
import cv2
# Define the translate function.
def translate(img, x, y):
    # Obtain the image size.
    (h, w) = img.shape[:2]

    # Define the translation matrix.
    M = np.float32([[1, 0, x], [0, 1, y]])

    # Use the OpenCV affine transformation function to implement the translation operation.
    shifted = cv2.warpAffine(img, M, (w, h))

    # Return the shifted image.
    return shifted

# Load and display the image.
im = cv2.imread('lena.jpg')
matshow("Orig", im)

# Translate the original image.
# 50 pixels down.
shifted = translate(im, 0, 50)
matshow("Shift1", shifted)
# 100 pixels left.
shifted = translate(im, -100, 0)
matshow("Shift2", shifted)
# 50 pixels right and 100 pixels down.
shifted = translate(im, 50, 100)
matshow("Shift3", shifted)
```

Output:

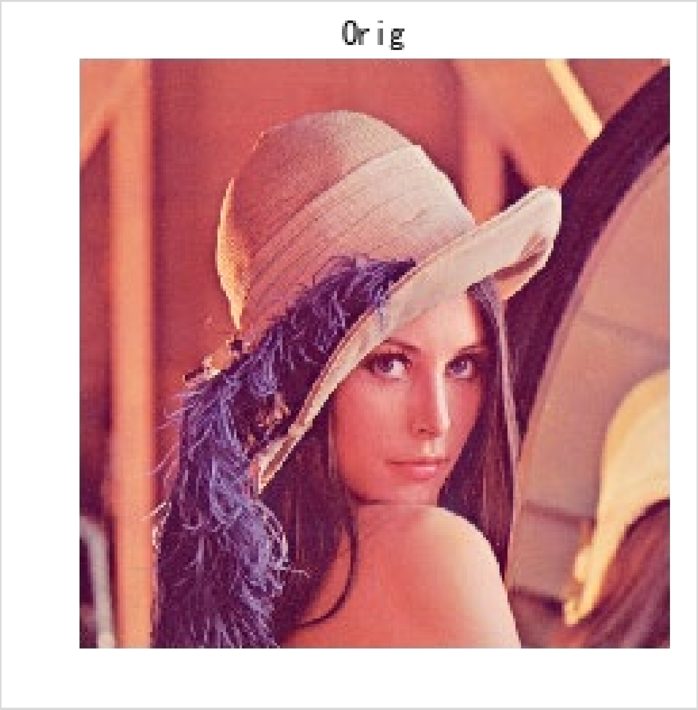


Figure 1-8 Original lena image

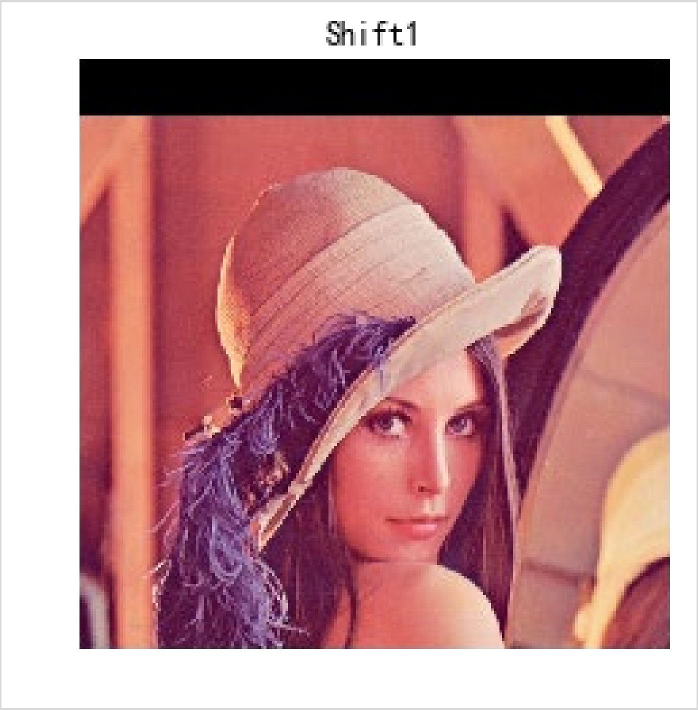


Figure 1-9 Move down a 50-pixel lena image

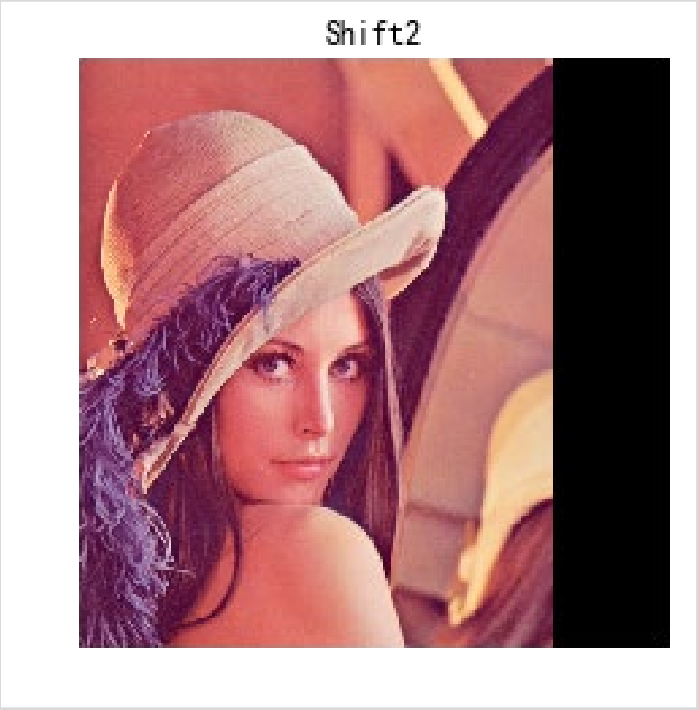


**Figure 1-10 Move the 100-pixel lena image to the left.**

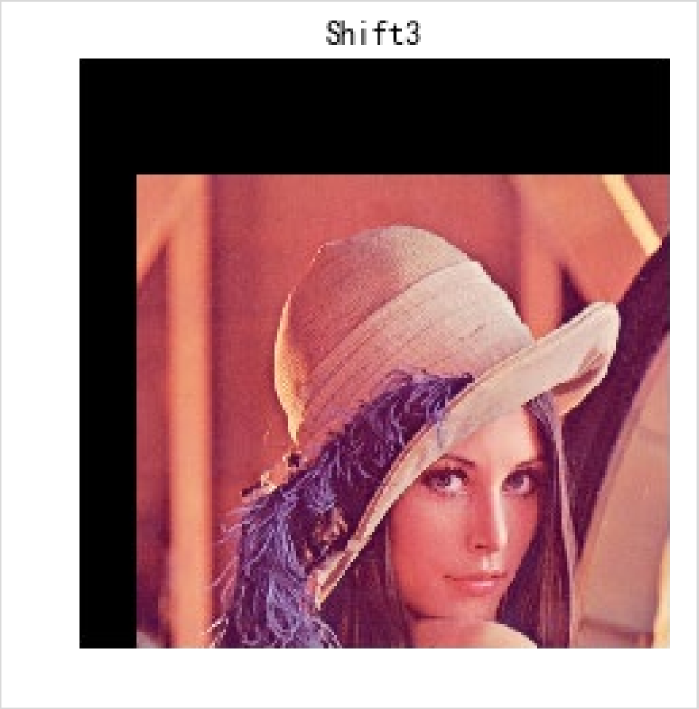


**Figure 1-11 Moves the image right by 50 pixels and moves down by 100 pixels.**

Step 2 rotation

```
import numpy as np
import cv2

# Define the rotate function.
def rotate(img, angle, center=None, scale=1.0):
    # Obtain the image size.
    (h, w) = img.shape[:2]

    # The missing value of the rotation center is the image center.
    if center is None:
        center = (w / 2, h / 2)

    # Invoke the function of calculating the rotation matrix.
    M = cv2.getRotationMatrix2D(center, angle, scale)

    # Use the OpenCV affine transformation function to implement the rotation operation.
    rotated = cv2.warpAffine(img, M, (w, h))

    # Return the rotated image.
    return rotated


im = cv2.imread('lena.jpg')
matshow("Orig", im)

# Rotate the original image.
# 45 degrees counterclockwise.
rotated = rotate(im, 45)
matshow("Rotate1", rotated)
# 20 degrees clockwise.
rotated = rotate(im, -20)
matshow("Rotate2", rotated)
# 90 degrees counterclockwise.
rotated = rotate(im, 90)
matshow("Rotate3", rotated)
```

Output:

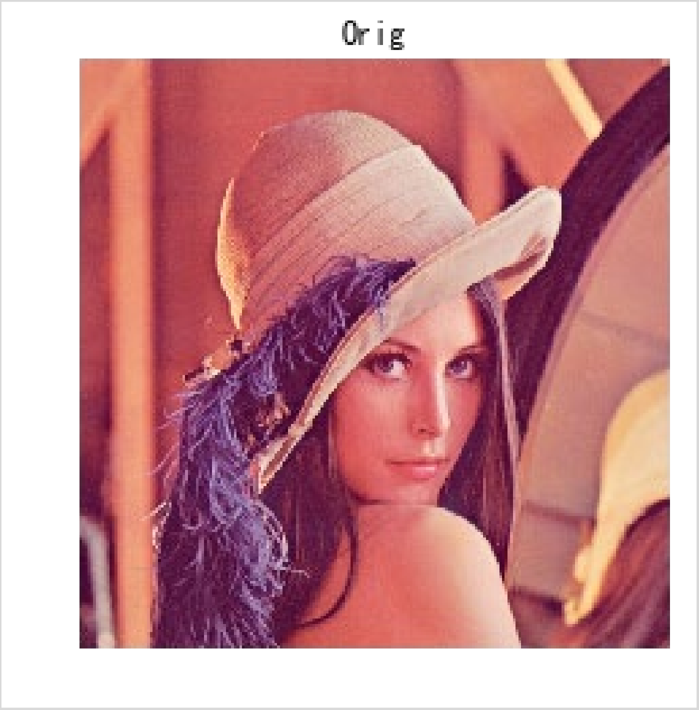


Figure 1-12 Original lena image

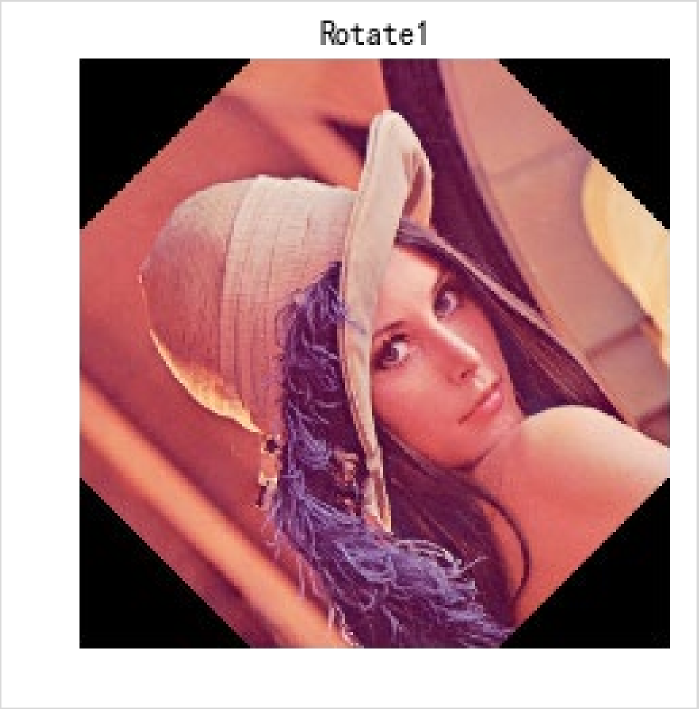


Figure 1-13 45 degrees counterclockwise lena image

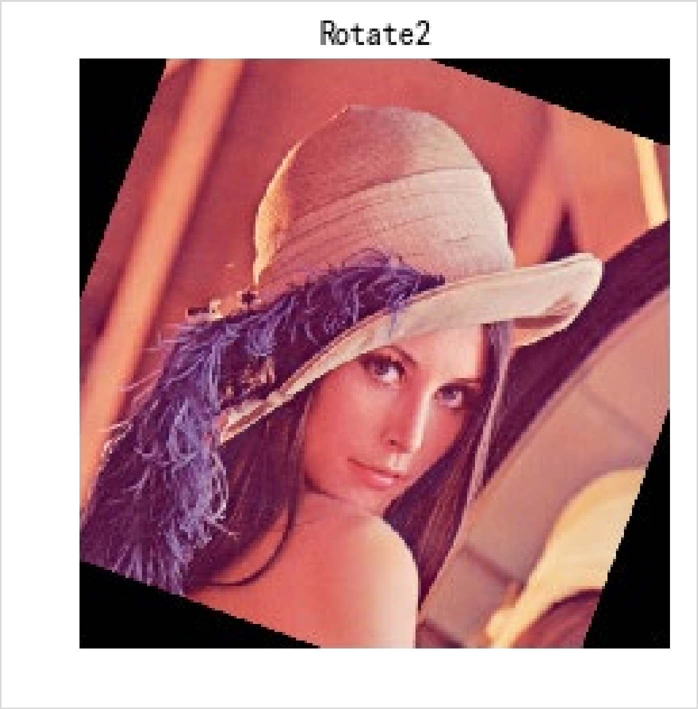


Figure 1-14 20 degrees clockwise lena image

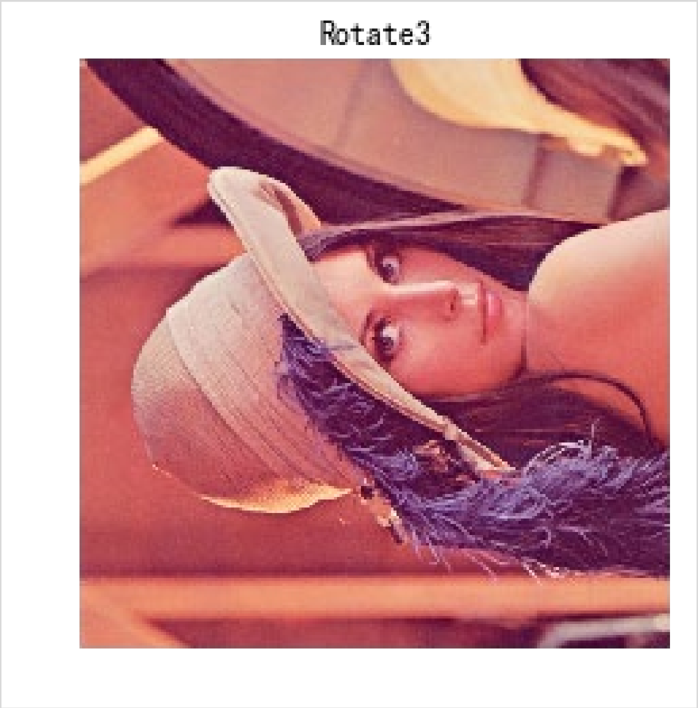


Figure 1-15 90 degrees counterclockwise lena image

Step 3 Mirroring

```
import numpy as np
import cv2

im = cv2.imread('lena.jpg')
matshow("orig", im)

# Perform vertical mirroring.
im_flip0 = cv2.flip(im, 0)
matshow("flip vertical", im_flip0)

im_flip1 = cv2.flip(im, 1)
# Perform horizontal mirroring.
matshow("flip horizontal", im_flip1)
```

Output:

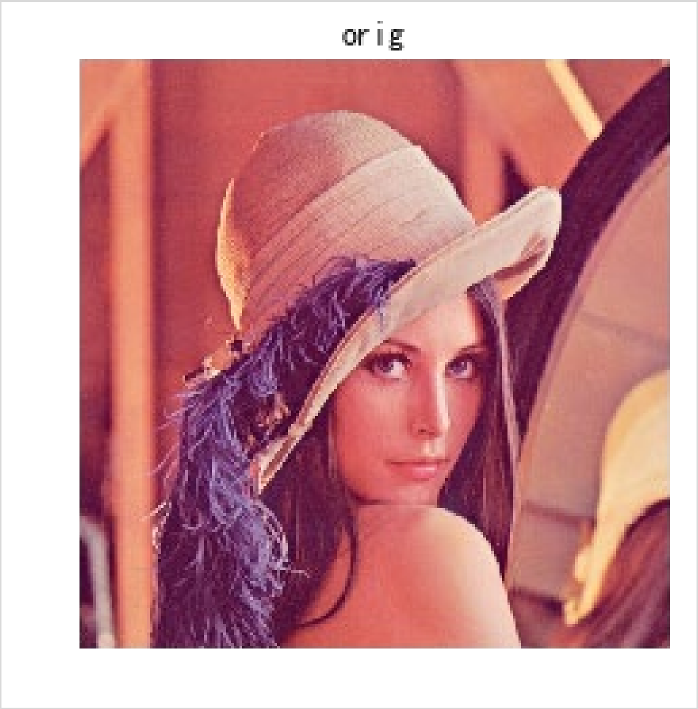


**Figure 1-16 Original lena image**

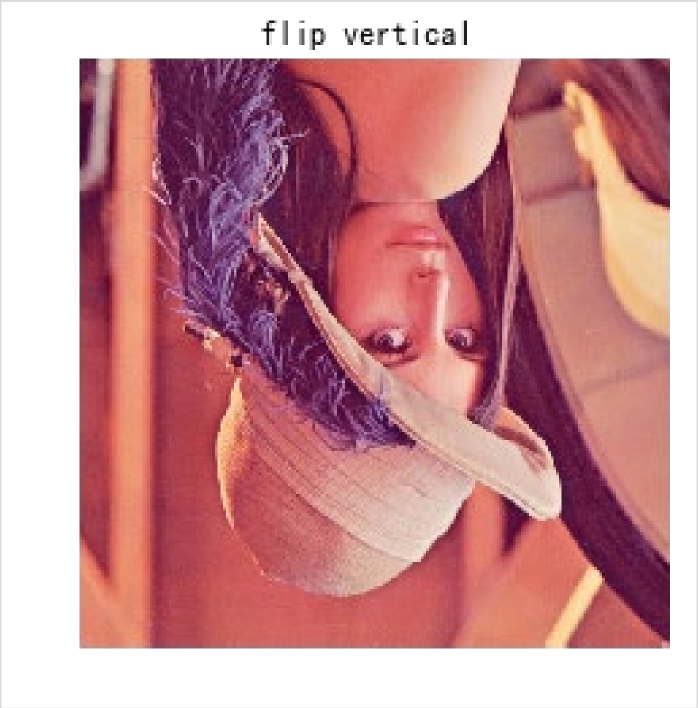


Figure 1-17 Vertical mirror lena image

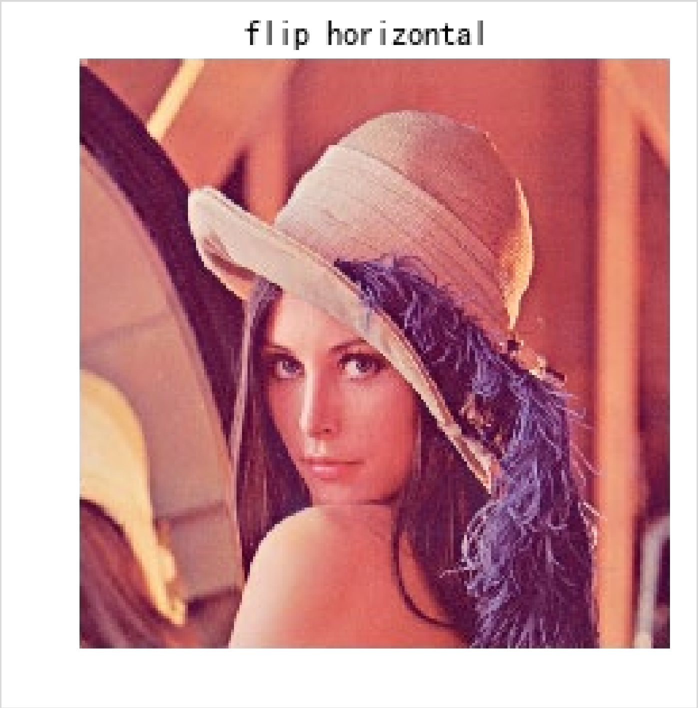


Figure 1-18 Horizontal mirror lena image

Step 4 Zoom

```
import numpy as np
import cv2

im = cv2.imread('lena.jpg')
matshow("orig", im)

# Obtain the image size.
(h, w) = im.shape[:2]

# Target size for scaling.
dst_size = (200, 300)

# Nearest interpolation
method = cv2.INTER_NEAREST

# Perform scaling.
resized = cv2.resize(im, dst_size, interpolation = method)
matshow("resized1", resized)

# Target size for scaling.
dst_size = (800, 600)

# Bilinear interpolation
method = cv2.INTER_LINEAR

# Perform scaling.
resized = cv2.resize(im, dst_size, interpolation = method)
matshow("resized2", resized)
```

Output:

**Figure 1-19 Original lena image**

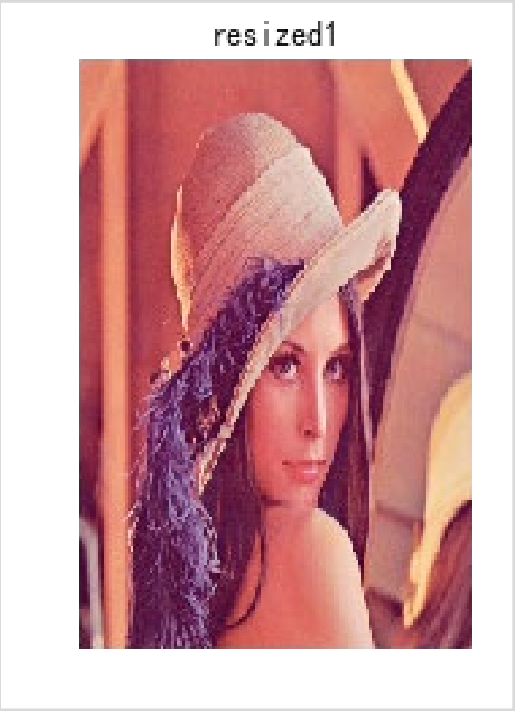


**Figure 1-20 nearest interpolation scaling lena image**

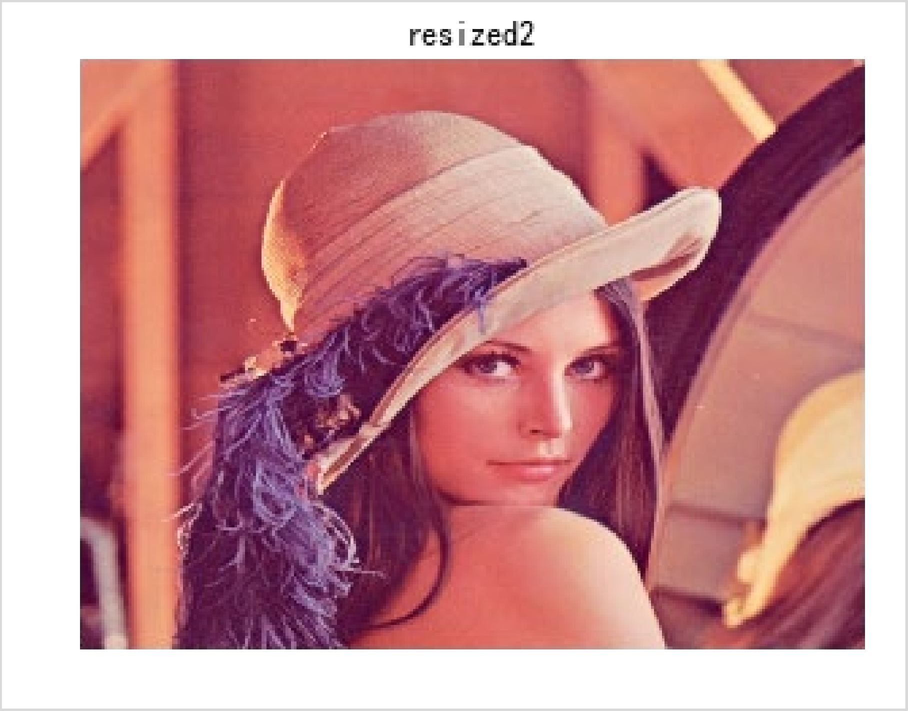


**Figure 1-21 Bilinear interpolation scaling lena image**

## 1.4.4 grayscale Transformation

Step 1 Grayscale Transformation. inversion, grayscale stretch, grayscale compression

```
# Define the linear grayscale transformation function.
# k > 1: Stretch the grayscale value.
# 0 < k < 1: Compress the grayscale value.
# k = -1, b = 255: Perform grayscale inversion.
def linear_trans(img, k, b=0):
    # Calculate the mapping table of linear grayscale changes.
    trans_list = [(np.float32(x)*k+b) for x in range(256)]
    # Convert the list to np.array.
    trans_table =np.array(trans_list)
    # Adjust the value out of the range [0,255] and set the data type to uint8.
    trans_table[trans_table>255] = 255
    trans_table[trans_table<0] = 0
    trans_table = np.round(trans_table).astype(np.uint8)
    # Use the look up table function in the OpenCV to change the image grayscale value.
    return cv2.LUT(img, trans_table)

im = cv2.imread('lena.jpg',0)
matshow('org', im)

# Inversion.
im_inversion = linear_trans(im, -1, 255)
matshow('inversion', im_inversion)
# Grayscale stretch.
im_stretch = linear_trans(im, 1.2)
matshow('graystretch', im_stretch)
# Grayscale compression.
im_compress = linear_trans(im, 0.8)
matshow('graycompress', im_compress)
```

Output:

Figure 1-22 Original lena grayscale image

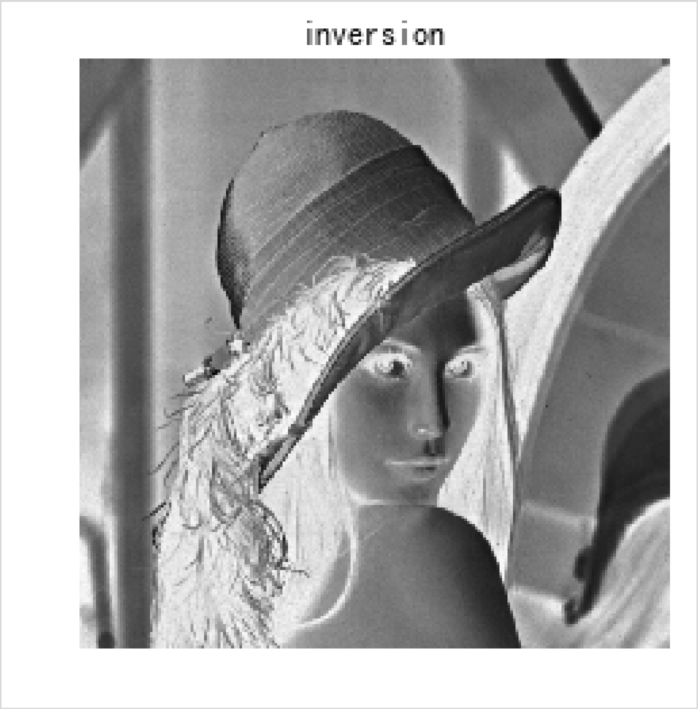


Figure 1-23 Flip the lena grayscale image.

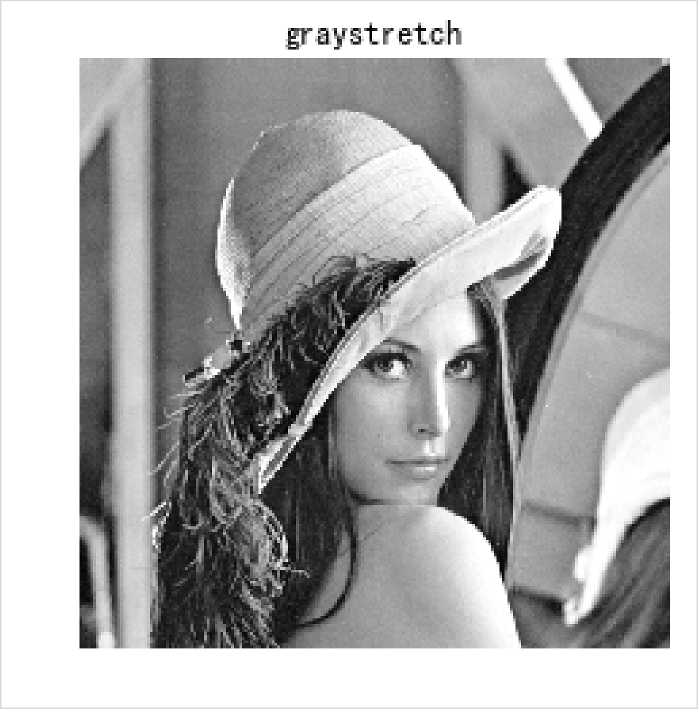


**Figure 1-24 Gray scale stretch lena gray scale chart**

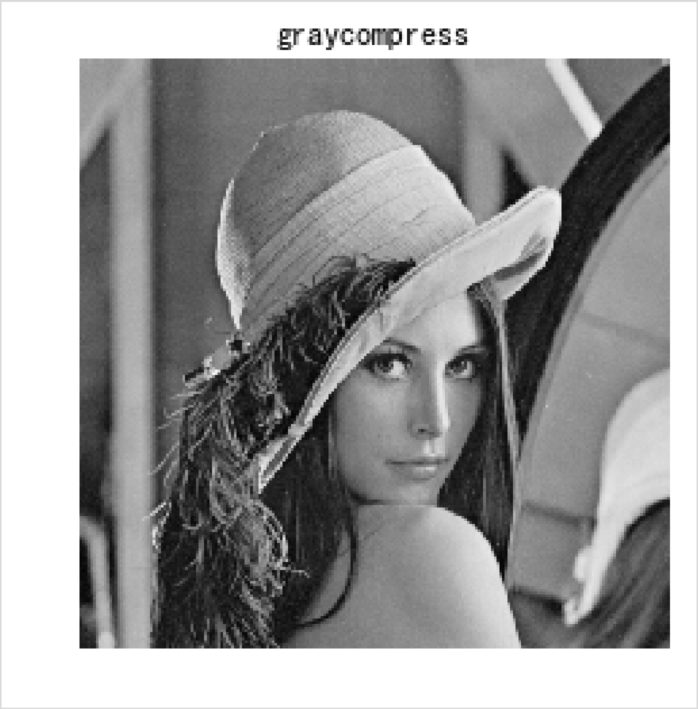


**Figure 1-25 Gray-scale compression lena grayscale image**

Step 2 gamma transformation

```
# Define the gamma transformation function.
def gamma_trans(img, gamma):
    # Firstly normalize the input to [0,1], perform the gamma function, and then restore the input to [0,255].
    gamma_list = [np.power(x / 255.0, gamma) * 255.0 for x in range(256)]
    # Convert list to np.array and set the data type to uint8.
    gamma_table = np.round(np.array(gamma_list)).astype(np.uint8)
    # Use the look up table function of the OpenCV to change the image grayscale value.
    return cv2.LUT(img, gamma_table)

im = cv2.imread('lena.jpg',0)
matshow('org', im)

# Use the gamma value 0.5 to stretch the shadow and compress the highlight.
im_gama05 = gamma_trans(im, 0.5)
matshow('gama0.5', im_gama05)
# Use the gamma value 2 to stretch the highlight and compress the shadow.
im_gama2 = gamma_trans(im, 2)
matshow('gama2', im_gama2)
```

Output:

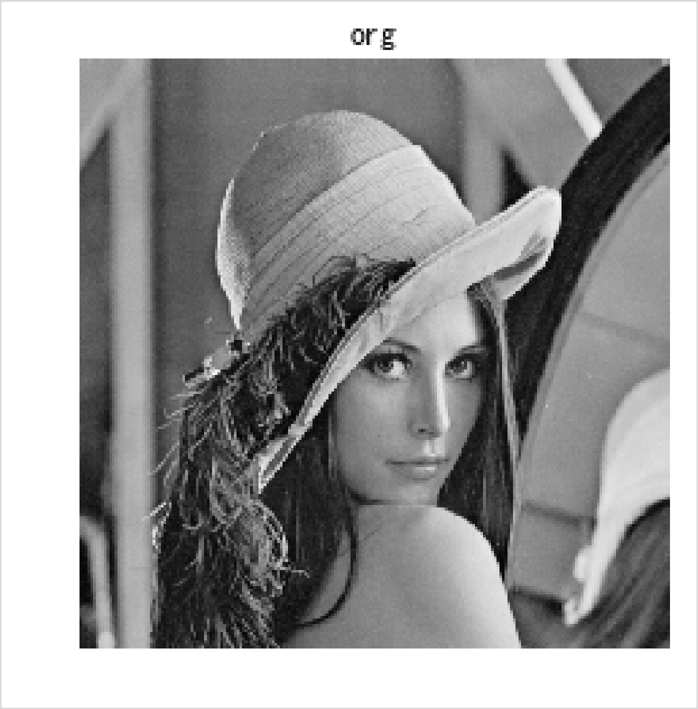


Figure 1-26 Original lena grayscale image

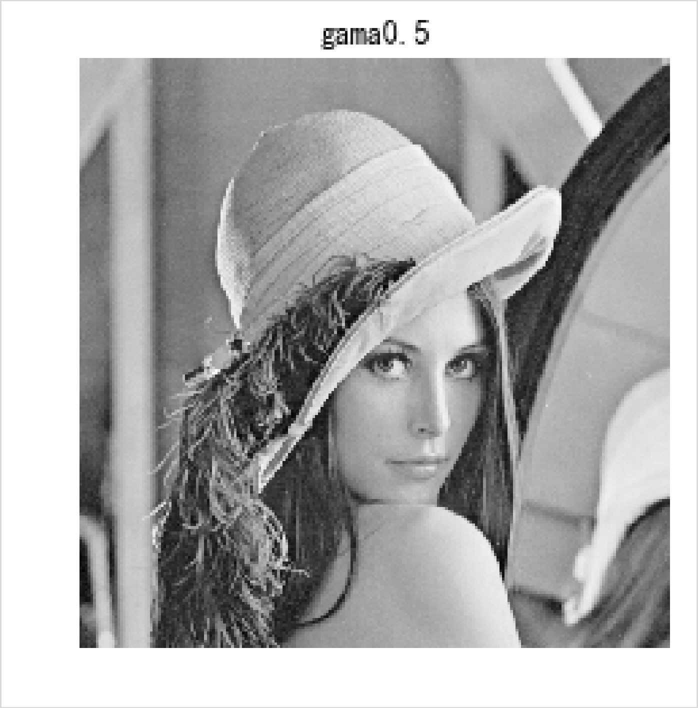


Figure 1-27 Gamma coefficient 0.5 lena grayscale chart

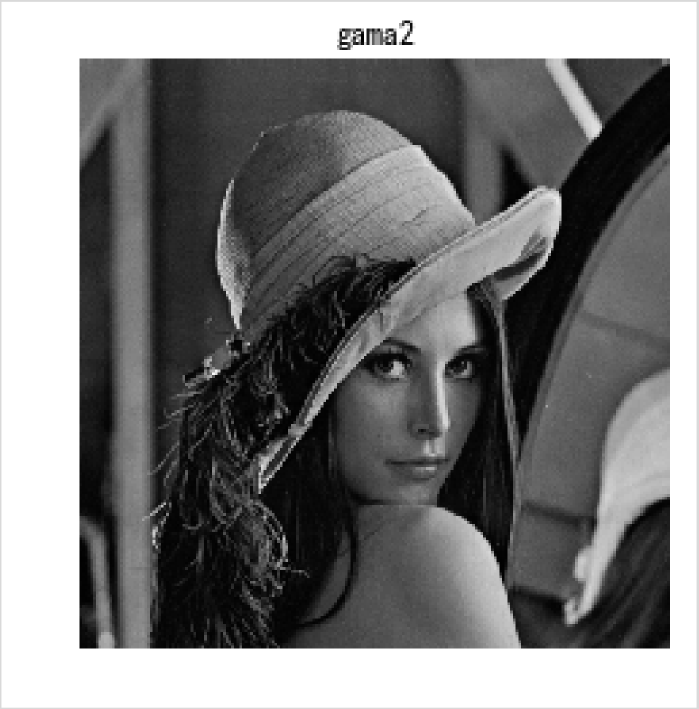


**Figure 1-28 Grayscale map with the gamma coefficient of 2 lena**

## 1.4.5 histogram

Step 1 Histogram display

```
from matplotlib import pyplot as plt
# Read and display the image.
im = cv2.imread("lena.jpg",0)
matshow('org', im)

# Draw a histogram for the grayscale image.
plt.hist(im.ravel(), 256, [0,256])
plt.show()
```

Output:

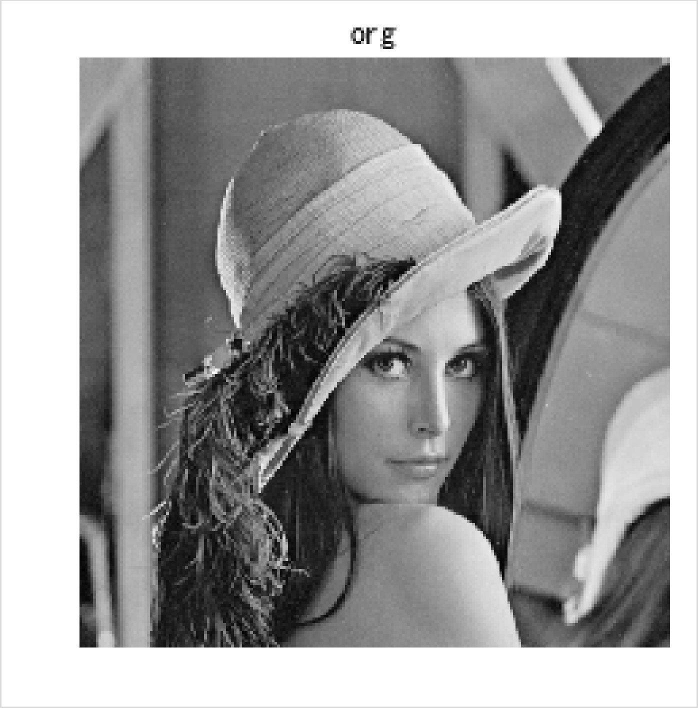


Figure 1-29 Original lena grayscale image

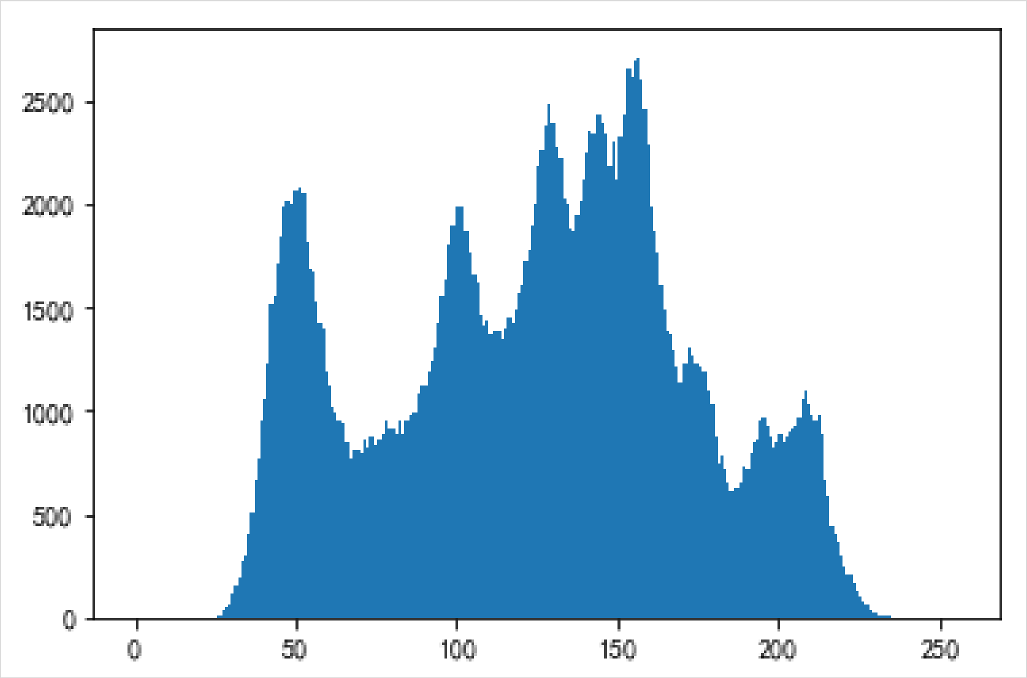


Figure 1-30 lena gray histogram

Step 2 histogram equalization

```
im = cv2.imread("lena.jpg",0)
matshow('org', im)

# Invoke the histogram equalization API of the OpenCV.
im_equ1 = cv2.equalizeHist(im)
```

```
matshow('equal', im_equ1)

# Display the histogram of the original image.
plt.subplot(2,1,1)
plt.hist(im.ravel(), 256, [0,256],label='org')
plt.legend()

# Display the histogram of the equalized image.
plt.subplot(2,1,2)
plt.hist(im_equ1.ravel(), 256, [0,256],label='equalize')
plt.legend()
plt.show()
```

Output:

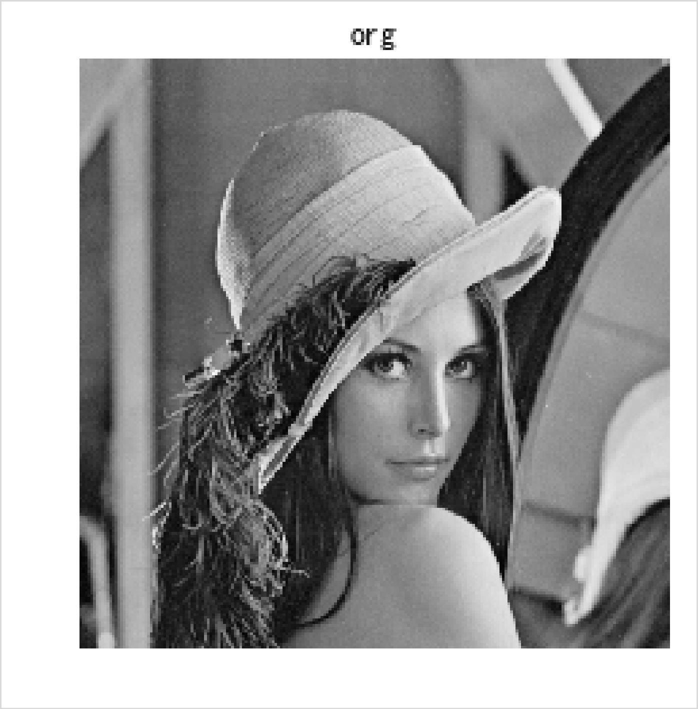


**Figure 1-31 Original lena grayscale image**

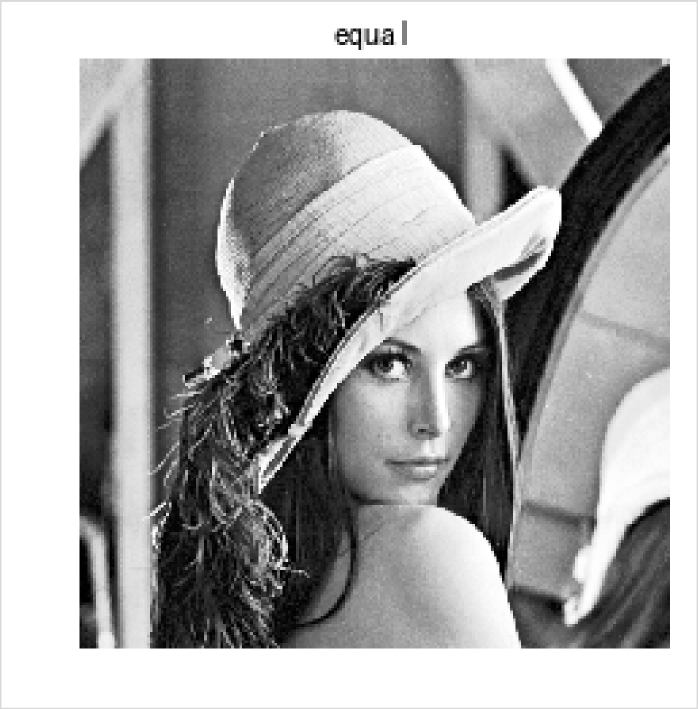


**Figure 1-32 Lena gray scale after histogram equalization**

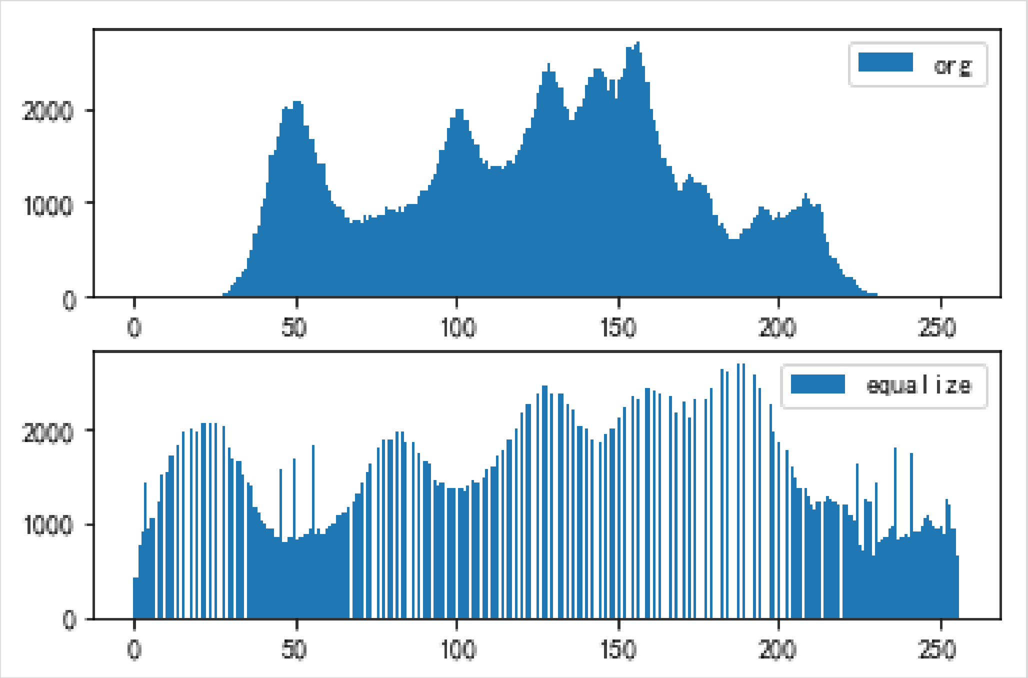


**Figure 1-33 Histogram comparison before and after equalization**

## 1.4.6 filtering

Step 1 median filtering

```
import cv2
import numpy as np

im = cv2.imread('lena.jpg')
matshow('org', im)

# Invoke the median fuzzy API of OpenCV.
im_medianblur = cv2.medianBlur(im, 5)

matshow('median_blur', im_medianblur)
```

Output:

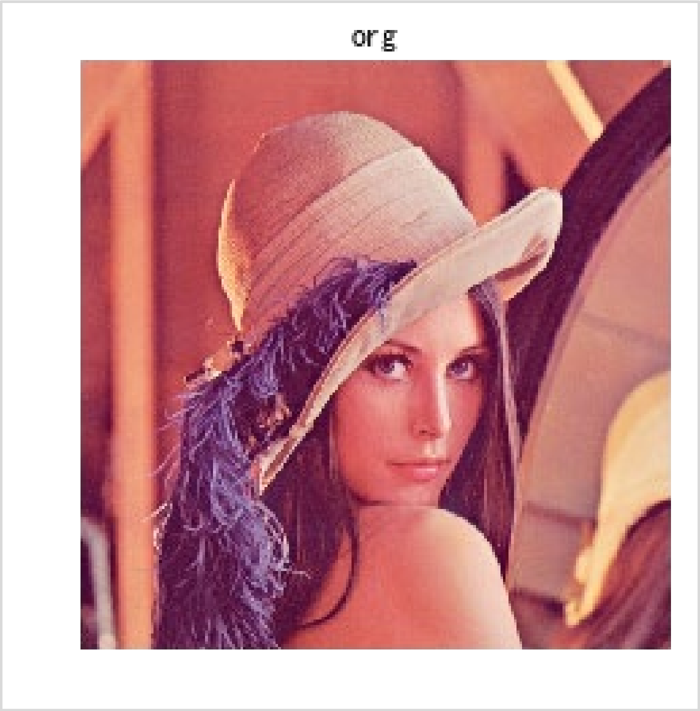


**Figure 1-34 Original lena image**

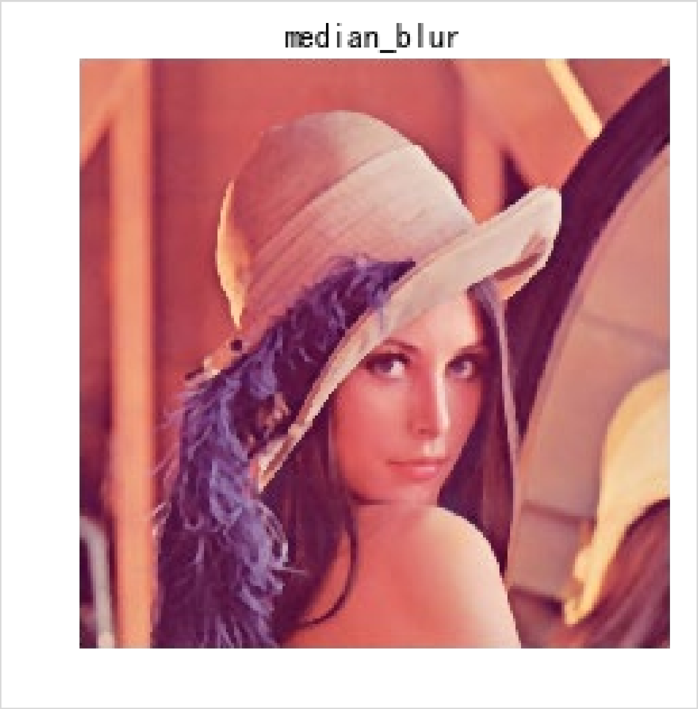


**Figure 1-35 Lena image after median filtering**

Step 2 mean filtering

```
# Method 1: Invoke the OpenCV API directly.
import cv2
import numpy as np

im = cv2.imread('lena.jpg')
matshow('org', im)

# Invoke the API for fuzzy average value of OpenCV.
im_meanblur1 = cv2.blur(im, (3, 3))

matshow('mean_blur_1', im_meanblur1)



# Method 2: Use mean operator and filter2D to customize filtering.
import cv2
import numpy as np

im = cv2.imread('lena.jpg')
matshow('org', im)
# mean operator
mean_blur = np.ones([3, 3], np.float32)/9

# Use filter2D to perform filtering.
im_meanblur2 = cv2.filter2D(im, -1, mean_blur)
matshow('mean_blur_2', im_meanblur2)
```

Output:

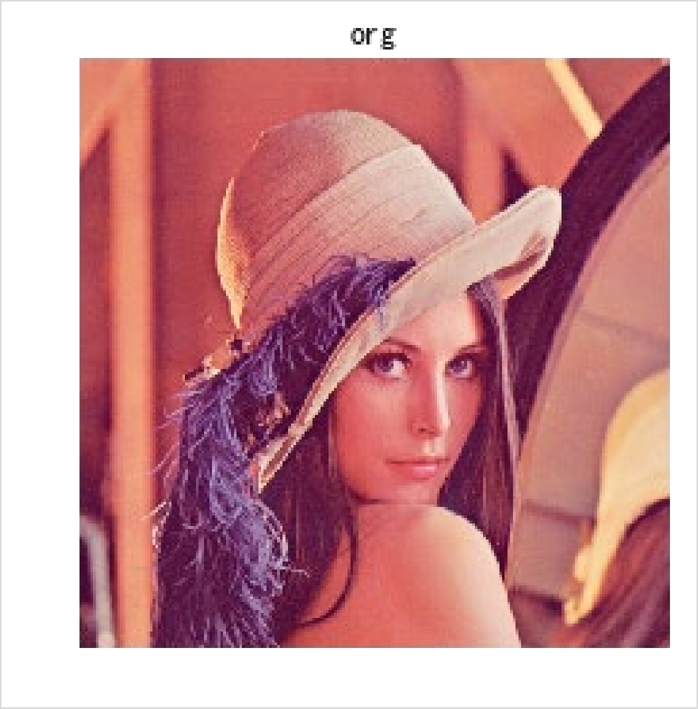


**Figure 1-36 Original lena image**

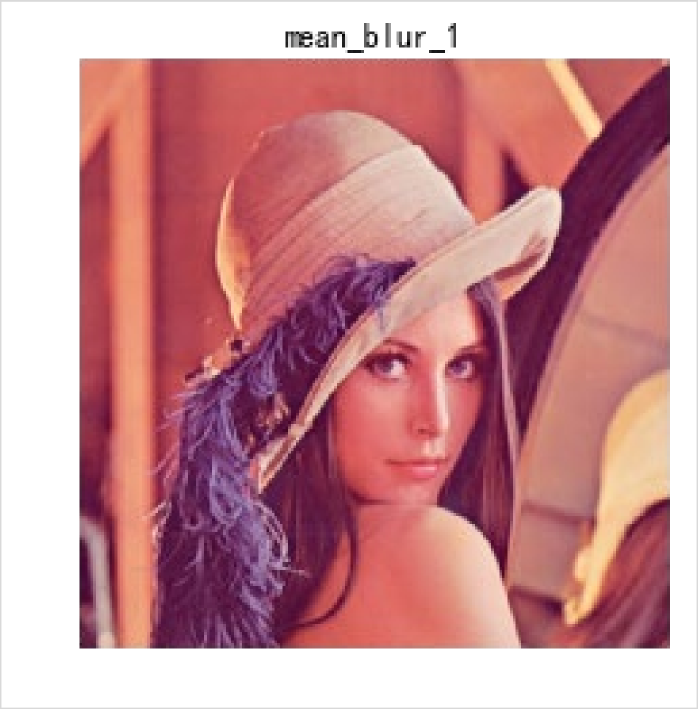


**Figure 1-37 Lena image after OpenCV mean filtering**

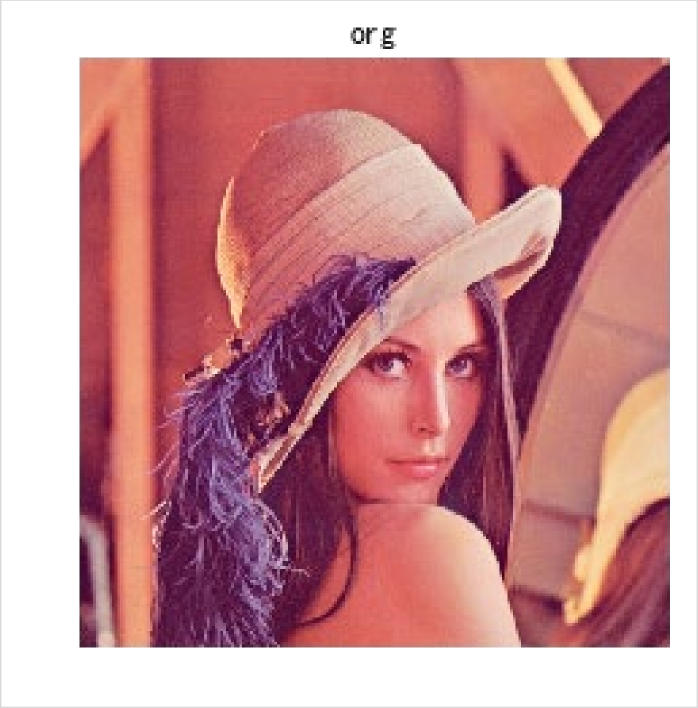


**Figure 1-38 Original lena image**

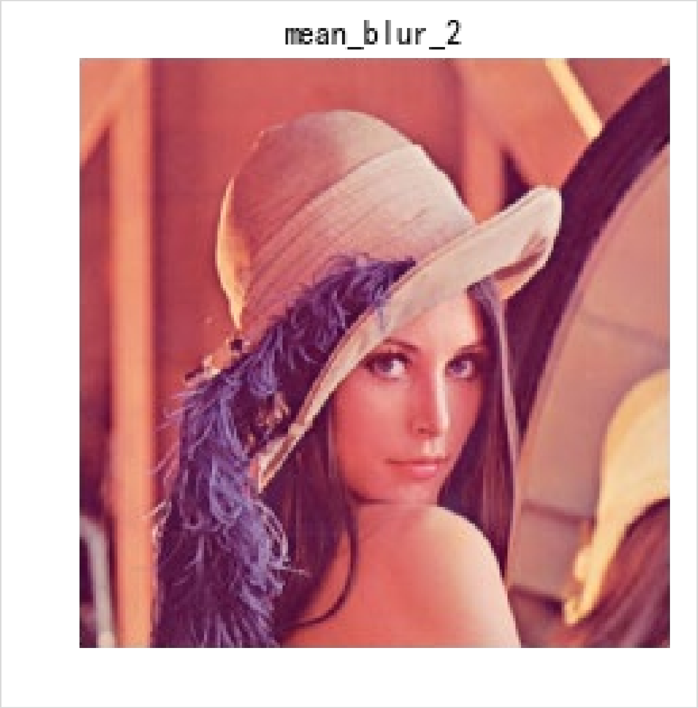


**Figure 1-39 Lena image after custom average filtering**

Step 3 Gaussian filtering

```
import cv2
import numpy as np

im = cv2.imread('lena.jpg')
```

```
matshow('org',im)

# Invoke the Gaussian filtering API of the OpenCV.
im_gaussianblur1 = cv2.GaussianBlur(im, (5, 5), 0)

matshow('gaussian_blur_1',im_gaussianblur1)

# Method 2: Use the Gaussian operator and filter2D to customize filtering operations.
import cv2
import numpy as np

im = cv2.imread('lena.jpg')
matshow('org',im)

# Gaussian operator
gaussian_blur = np.array([
        [1,4,7,4,1],
        [4,16,26,16,4],
        [7,26,41,26,7],
        [4,16,26,16,4],
        [1,4,7,4,1]], np.float32)/273

# # Use filter2D to perform filtering.
im_gaussianblur2 = cv2.filter2D(im,-1,gaussian_blur)
matshow('gaussian_blur_2',im_gaussianblur2)
```

Output:

Figure 1-40 Original lena image

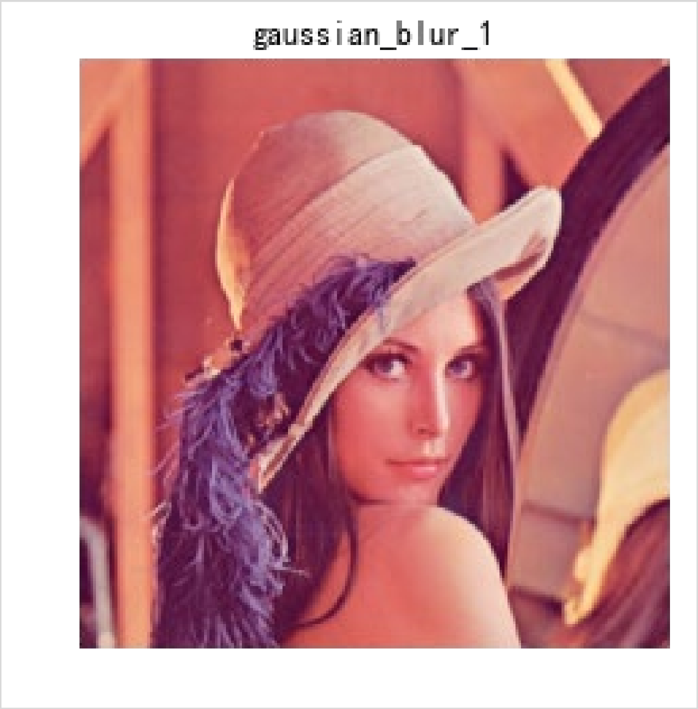


Figure 1-41 Lena image after OpenCV Gaussian filtering is used

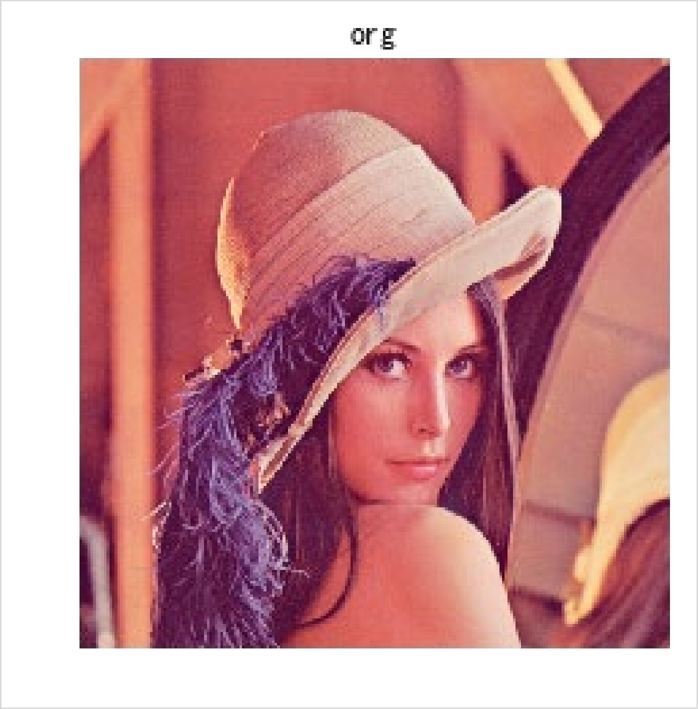


**Figure 1-42 Original lena image**

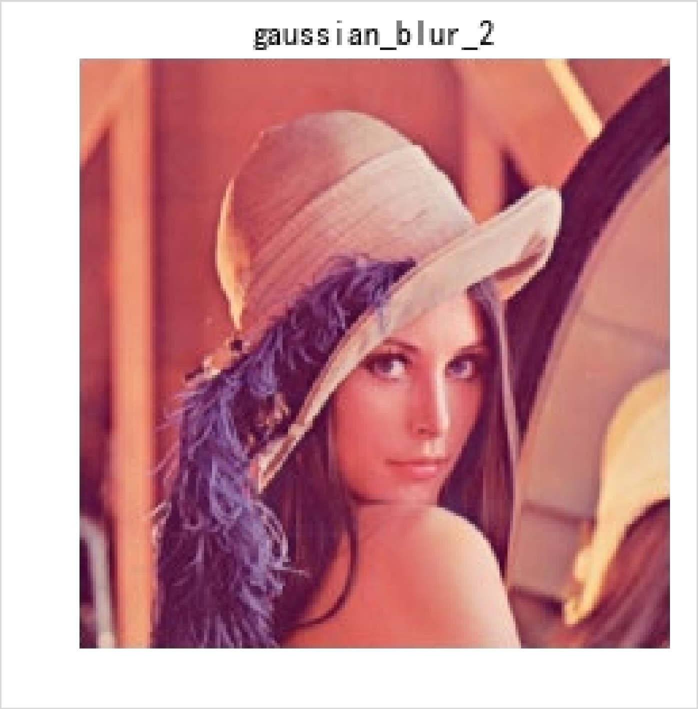


**Figure 1-43 Lena image after user-defined Gaussian filtering**

Step 4 sharpening

```
im = cv2.imread('lena.jpg')
matshow('org',im)
# Sharpening operator 1.
sharpen_1 = np.array([
            [-1,-1,-1],
            [-1,9,-1],
            [-1,-1,-1]])
# Use filter2D to perform filtering.
im_sharpen1 = cv2.filter2D(im,-1,sharpen_1)
matshow('sharpen_1',im_sharpen1)

# Sharpening operator 2.
sharpen_2 = np.array([
            [0,-1,0],
            [-1,8,-1],
            [0,1,0]])/4.0

# Use filter2D to perform filtering.
im_sharpen2 = cv2.filter2D(im,-1,sharpen_2)
matshow('sharpen_2',im_sharpen2)
```

Output:

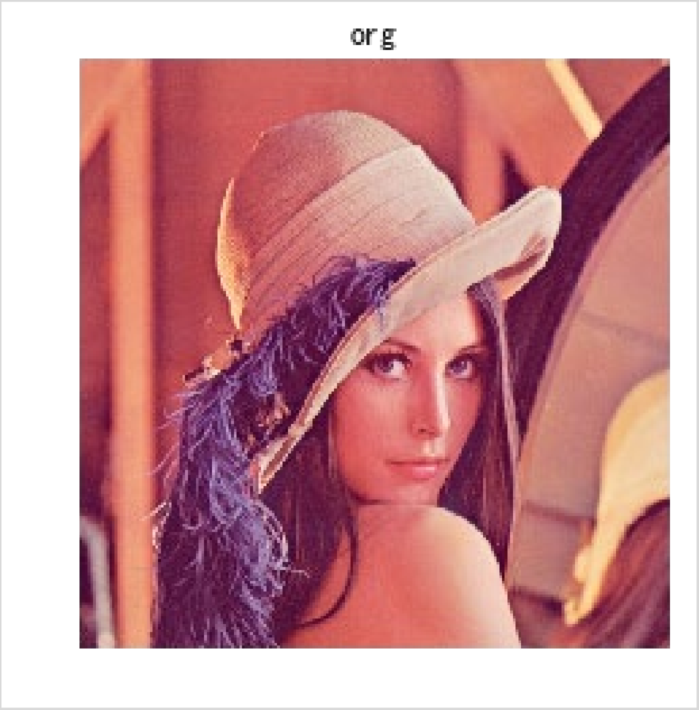


**Figure 1-44 Original lena image**

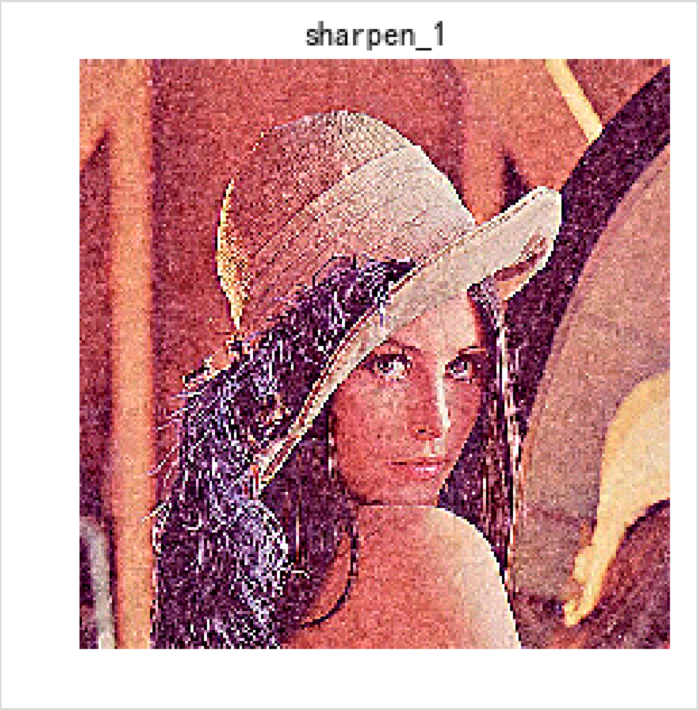


**Figure 1-45 Sharpening lena image 1**

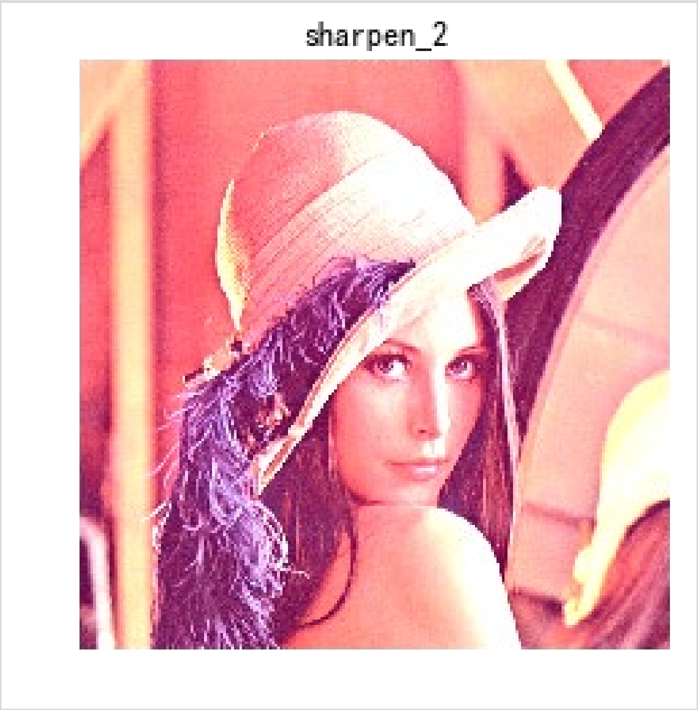


**Figure 1-46 Sharpening lena image 2**

## 1.5 Experiment Summary

This section describes how to use the OpenCV image processing library to preprocess images in Python. In this experiment, the OpenCV image processing library is used to implement basic image preprocessing operations, including color space conversion, coordinate transformation, grayscale transformation, histogram transformation, and image filtering. This section can deepen the perception of the image preprocessing technology and provide practical operation guidance for using the technology.

# 2 HUAWEI CLOUD EI Image Tag Service

## 2.1 Introduction to the Experiment

Image recognition is a technology that uses a computer to process, analyze, and understand images to identify objects in different modes. Image recognition is available through open application programming interfaces (APIs). You can obtain the prediction results by accessing and invoking the APIs in real time. The APIs help you collect key data automatically and build an intelligent service system, thereby improving service efficiency.

Natural images have rich semantic content. An image contains multiple tags. HUAWEI CLOUD Image tags services can identify more than 3000 objects and more than 20,000 scenes and concept tags, making certain applications such as intelligent album management, photo search and classification, and scenario-based content or object-based ad recommendation more accurate.

In the information age, people are used to taking photos with their mobile phones. However, the information age has also brought about an explosion of information, and if not properly organized, people's electronic devices may have thousands of photographs, which are difficult to clear up.

There are a lot of software on the market for making electronic albums, but there are some limitations and some are expensive. By combining AI APIs and Python functions provided by HUAWEI CLOUD EI, you can customize your desired albums.

This lab describes how to use the image recognition service of HUAWEI CLOUD to implement simple electronic album arrangement.

## 2.2 Objective

This exercise describes how to use image tagging services to tag images. Currently, Huawei public cloud provides the RESTful API of image recognition and the SDK based on Python. This exercise will guide trainees to understand and master how to use Python to use the image tag service to intelligently arrange albums.

# 2.3 Lab APIs

## 2.3.1 REST APIs

HUAWEI CLOUD APIs comply with RESTful API design specifications. Representational State Transfer (REST) allocates Uniform Resource Identifiers (URIs) to dispersed resources so that the resources can be located. Applications on clients use Uniform Resource Locators (URLs) to obtain the resources.

## 2.3.2 REST API Request/Response Structure

A RESTful API request/response consists of the following five parts:

- Request URL

The URL format is as follows: https://***Endpoint/uri***. The parameters in the URL are described in URL.

**Table 2-1 URL parameter description**

| **Parameter** | **Description** |
|---|---|
| Endpoint | Web service entrance URL. Obtain this value from Regions and Endpoints.<br>Endpoint image.cn-north-4.myhuaweicloud.com corresponding to the image recognition service is used by all service APIs. |
| uri | Resource path, that is, the API access path. Obtain the value from the URI of the API, for example, /v1.0/ais/subscribe. |

- Request header

The request header consists of two parts: HTTP method and optional additional request header field (such as the field required by a specified URI and HTTP method).

Table 2-2 describes the request methods supported by RESTful APIs.

**Table 2-2 Request method description**

| Method | Description |
|---|---|
| GET | Requests the server to return specified resources. |
| PUT | Requests the server to update specified resources. |
| POST | Requests the server to add resources or perform a special operation. |
| DELETE | Requests the server to delete specified resources, for example, objects. |
| PATCH | Requests the server to update partial content of a specified resource.<br>If a target resource does not exist, PATCH may create a resource. |

- Request body

A request body is generally sent in a structured format (for example, JSON or XML), corresponding to Content-type in the request header, and is used to transfer content except the request header. If a request body contains a parameter in Chinese, the parameter must be coded in UTF-8 mode.

- Response header

A response header contains two parts: status code and additional response header field.

Status code, including success codes 2xx and error codes 4xx or 5xx. Additional response header field, such as the field required by the response supporting a request (the field in the Content-type response header).

- Response body

A response body is generally returned in a structured format (for example, JSON or XML), and is used to transfer content except the response header. When a service request is successfully sent, the service request result is returned. When a service request fails to be sent, an error code is returned. Request Initiation Methods

There are three methods to initiate constructed requests, including:

- cURL

cURL is a command line tool, which can be used to perform URL operations and transfer information. cURL functions as an HTTP client can send HTTP requests to the server and receive responses. cURL is applicable to API debugging.

- Code

You can invoke APIs by coding to assemble, send, and process requests.

Mozilla and Google provide graphical browser plug-ins for REST clients to send and process requests.

## 2.3.3 Image Tagging API

**Function overview:**

Natural images have rich semantic meanings because one image contains various tags. Image tagging can recognize hundreds of scenarios and thousands of objects and their properties in natural images, making intelligent album management, image retrieval and classification, and scenario- or object-based advertising more intuitive. After the image to be processed is uploaded, image tagging will return the tag and confidence score.

URI

URI format: POST /v1.0/image/tagging

Request

**Table 2-3 Request parameter description**

| Parameter | Mandatory or Optional | Type | Description |
|---|---|---|---|
| image | Set either | String | Image data, which is encoded based on Base64. |

| | | | |
|---|---|---|---|
| | this parameter or url. | | The size of data encoded based on Base64 cannot exceed 10 MB. The image resolution of the short edges must be greater than or equal to 15 pixels, and that of the long edges cannot exceed 4096 pixels. The supported image formats include JPG, PNG, and BMP. |
| url | Set either this parameter or image. | String | URL of the image file. Currently, this URL can be accessed by temporarily authorization on HUAWEI CLOUD OBS or anonymous and public authorization. |
| language | Optional | String | Language type of the returned tag. The default value is zh, which indicates Chinese. 'en' can be chosen as English. |
| limit | Optional | Integer | Maximum number of tags that can be returned. The default value is -1, indicating that all tags are returned. |
| threshold | Optional | Float | Threshold (0 to 100) of the confidence score. The tags whose confidence score is lower than the threshold will not be returned. The default value is 0. |

Response

**Table 2-4 Response parameter description**

| Parameter | Type | Description |
|---|---|---|
| result | JSON | Content of the image tag returned when the invoking succeeds.<br>The parameter is not included when the API invoking fails. |
| tags | List | List of tags. |
| confidence | Float | Confidence score ranging from 0 to 100. |
| tag | String | Tag name. |
| error_code | String | Error code returned when the invoking fails. For details, see **Error Codes.**<br>The parameter is not included when the API invoking succeeds. |
| error_msg | String | Error message returned when the API invoking fails.<br>The parameter is not included when the API invoking succeeds. |

**Returned values**

- Normal

200

- Failed

Table 2-5 Returned Value parameter description

| Returned Value | Description |
|---|---|
| 400 | The request cannot be understood by the server due to malformed syntax. A client shall not submit the request again unless the request is modified.<br>The request parameters are incorrect. |
| 401 | The request requires user authentication. |
| 403 | No permission to perform this operation. |
| 404 | The request failed because the requested resource could not be found on the server. |
| 500 | The server encountered an unexpected fault which prevented it from processing the request. |

## 2.4 Procedure

In this experiment, you need to download the SDK for image recognition from the HUAWEI CLOUD service platform and use either of the following two methods to access the SDK. One method is to submit a RESTful service request by invoking the underlying APIs encapsulated by the SDK based on the AK/SK for identity authentication. The other method is to simulate the browser to submit a RESTful request by obtaining the user's token information. The procedure is as follows.Procedures:

### 2.4.1 Applying for a Service

Step 1 Open the HUAWEI CLOUD official website. https://www.huaweicloud.com/en-us/

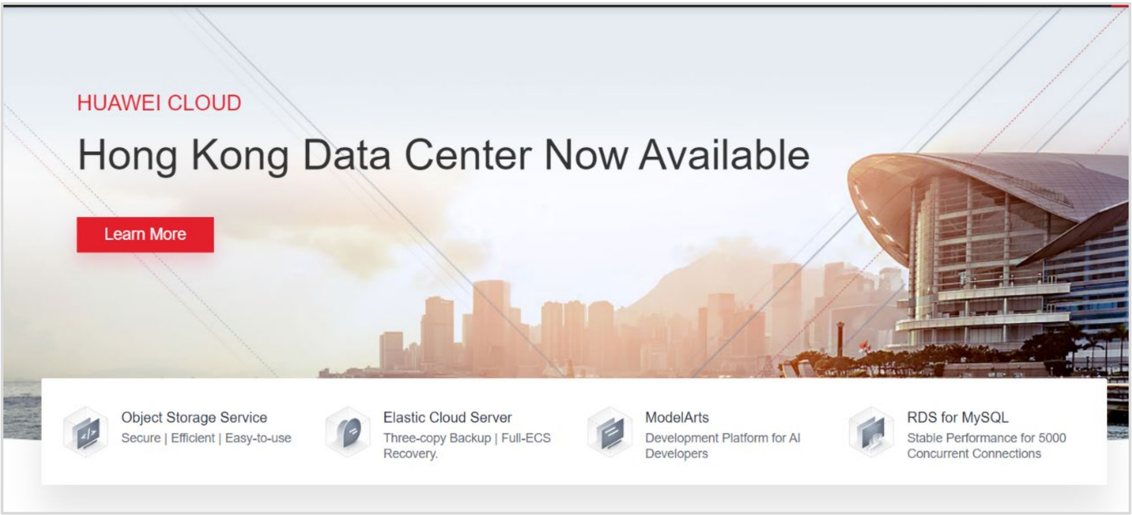

HUAWEI CLOUD
Hong Kong Data Center Now Available
Learn More
Object Storage Service
Secure | Efficient | Easy-to-use
Elastic Cloud Server
Three-copy Backup | Full-ECS Recovery.
ModelArts
Development Platform for AI Developers
RDS for MySQL
Stable Performance for 5000 Concurrent Connections

**Figure 2-1 HUAWEI CLOUD official website**

Step 2 Log in to the system using a HUAWEI CLOUD account and choose image recognition.

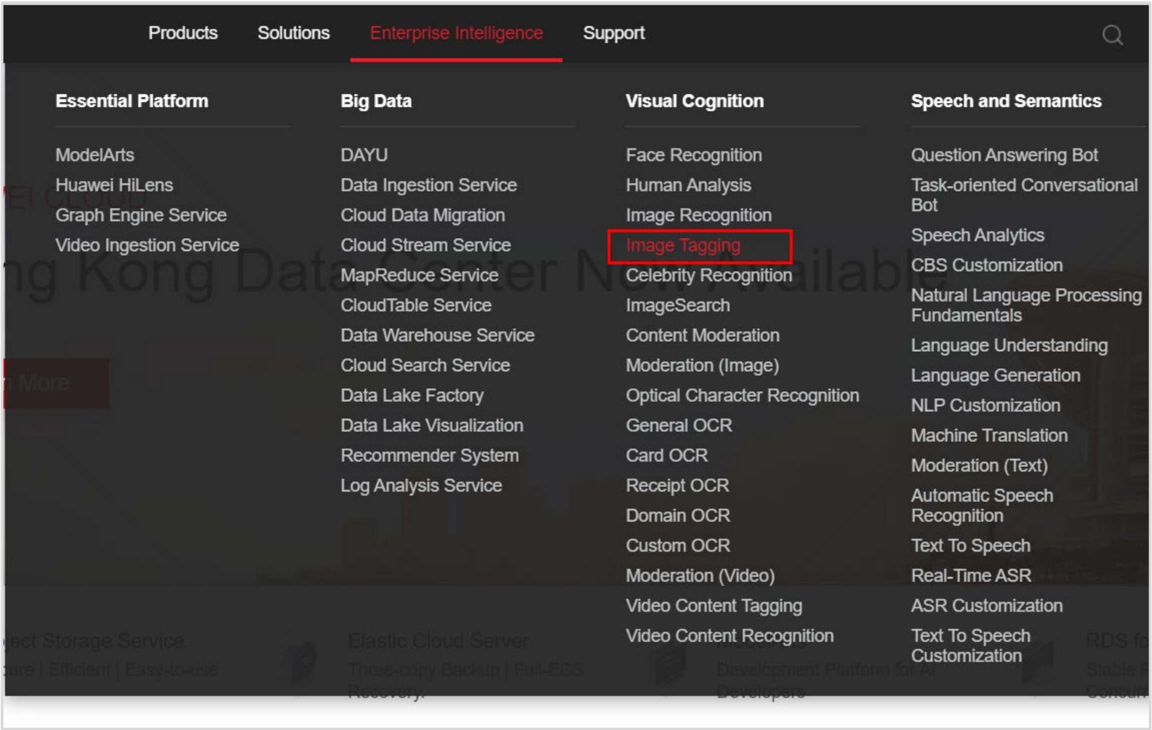


**Figure 2-2 Image label under EI**

Step 3 Click Use Now:

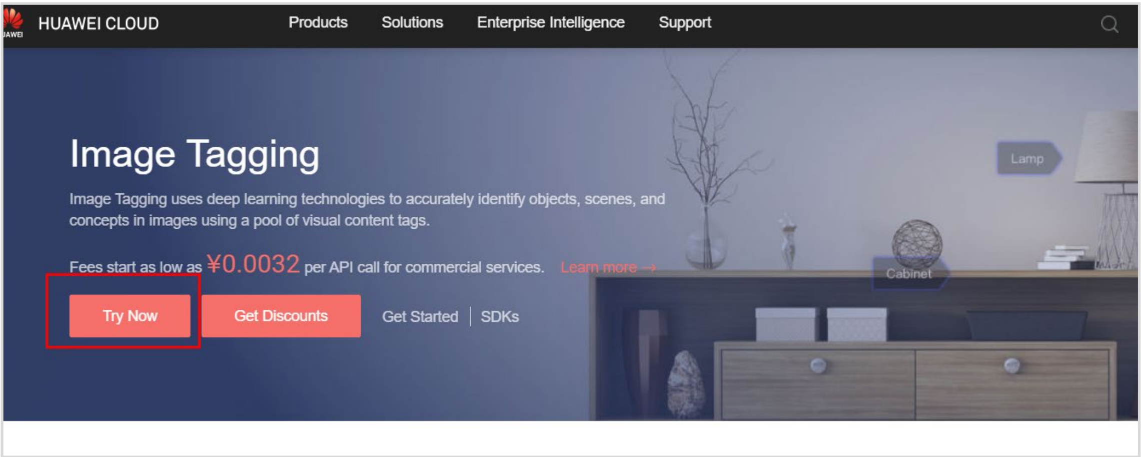


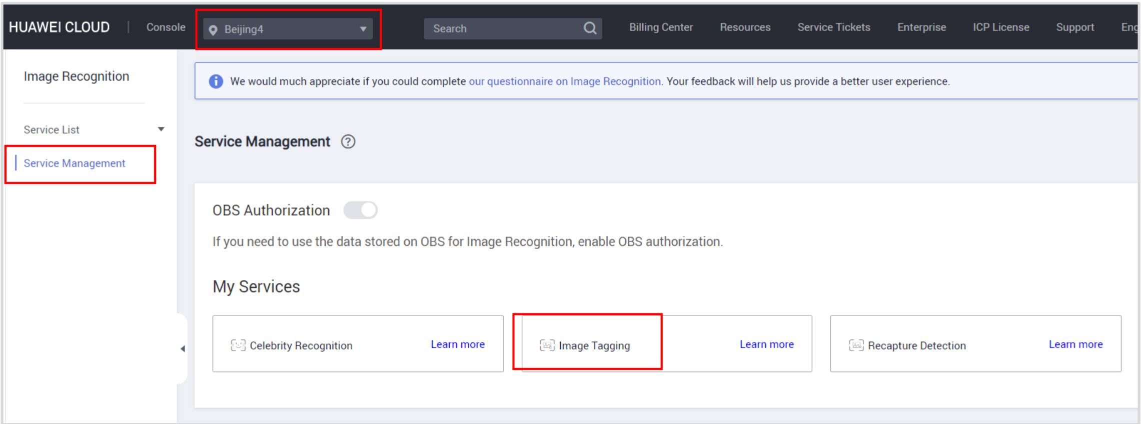
Image Tagging

Image Tagging uses deep learning technologies to accurately identify objects, scenes, and concepts in images using a pool of visual content tags.

Fees start as low as ¥0.0032 per API call for commercial services. Learn more →

**Figure 2-3 image recognition main window**

Step 4 Select Beijing 4 and enable the corresponding service. In this experiment, you need to enable Image Tag.



We would much appreciate if you could complete our questionnaire on Image Recognition. Your feedback will help us provide a better user experience.

Service Management

OBS Authorization

If you need to use the data stored on OBS for Image Recognition, enable OBS authorization.

My Services

**Figure 2-4 Provisioning a Service**

## 2.4.2 (Optional) Downloading the image recognition SDK

In this lab, the SDK is used as a service, which has been integrated in subsequent data sets. You can choose whether to use the SDK to set up an environment independently.

Step 1 Downloading the image recognition SDK Software Package and Documents

Link: https://developer.huaweicloud.com/en-us/sdk?IMAGE

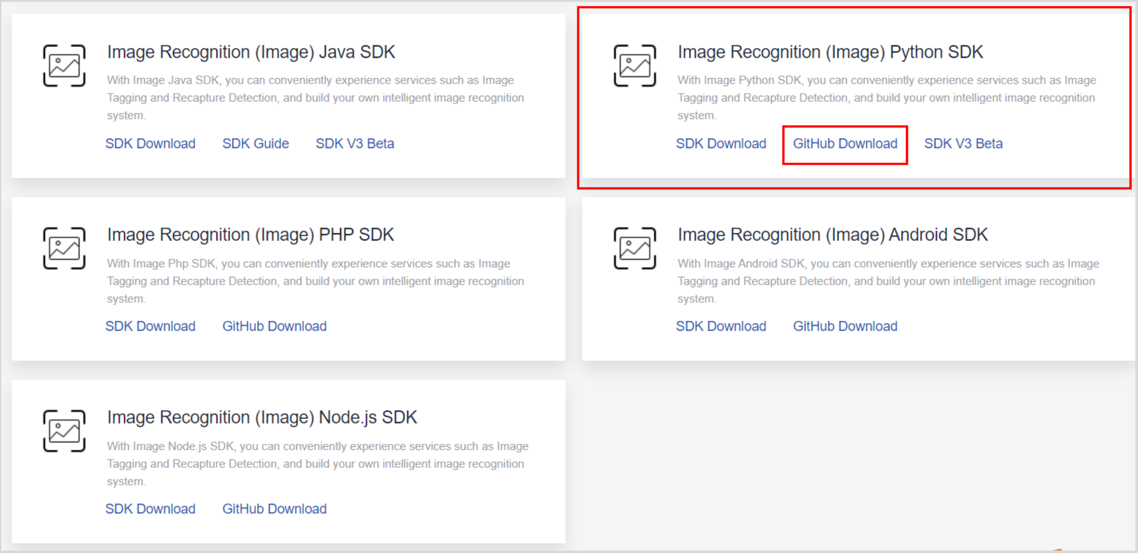


Image Recognition (Image) Java SDK

With Image Java SDK, you can conveniently experience services such as Image Tagging and Recapture Detection, and build your own intelligent image recognition system.

Image Recognition (Image) Python SDK

With Image Python SDK, you can conveniently experience services such as Image Tagging and Recapture Detection, and build your own intelligent image recognition system.

Image Recognition (Image) PHP SDK

With Image Php SDK, you can conveniently experience services such as Image Tagging and Recapture Detection, and build your own intelligent image recognition system.

Image Recognition (Image) Android SDK

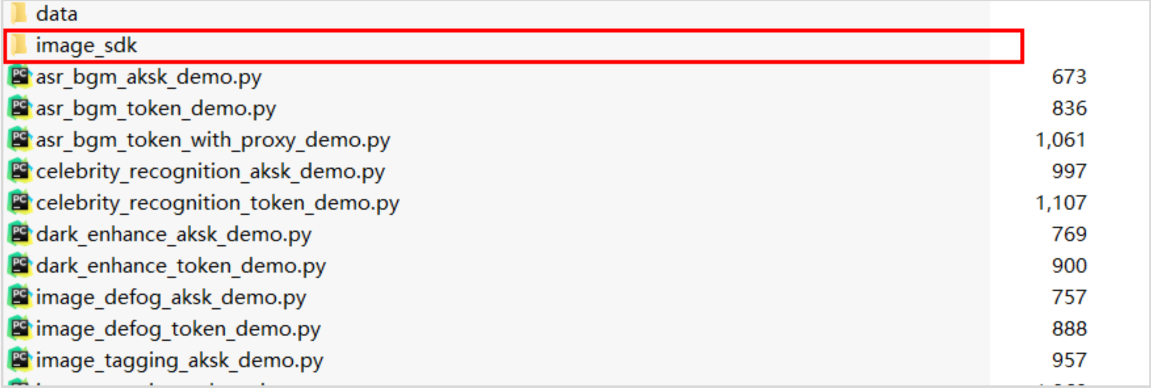
With Image Android SDK, you can conveniently experience services such as Image Tagging and Recapture Detection, and build your own intelligent image recognition system.

Image Recognition (Image) Node.js SDK

With Image Node.js SDK, you can conveniently experience services such as Image Tagging and Recapture Detection, and build your own intelligent image recognition system.

**Figure 2-5 HUAWEI CLOUD SDK**

Step 2 Decompress the image_sdk folder in the package to the project folder.



| | |
|---|---|
| data | |
| image_sdk | |
| asr_bgm_aksk_demo.py | 673 |
| asr_bgm_token_demo.py | 836 |
| asr_bgm_token_with_proxy_demo.py | 1,061 |
| celebrity_recognition_aksk_demo.py | 997 |
| celebrity_recognition_token_demo.py | 1,107 |
| dark_enhance_aksk_demo.py | 769 |
| dark_enhance_token_demo.py | 900 |
| image_defog_aksk_demo.py | 757 |
| image_defog_token_demo.py | 888 |
| image_tagging_aksk_demo.py | 957 |

**Figure 2-6 Move to Project Folder**

## 2.4.3 Use AK/SK to perform image tag management. (Skip this step if you alrealy have ak/sk)

Obtain the access key (AK) and secret access key (SK). The AK and SK are the keys used to access your own account. The AK and SK are required for calling image recognition APIs. If you have obtained the AK and SK, skip this step.

Step 1 Open the HUAWEI CLOUD official website. https://www.huaweicloud.com/en-us/Log in to the console.

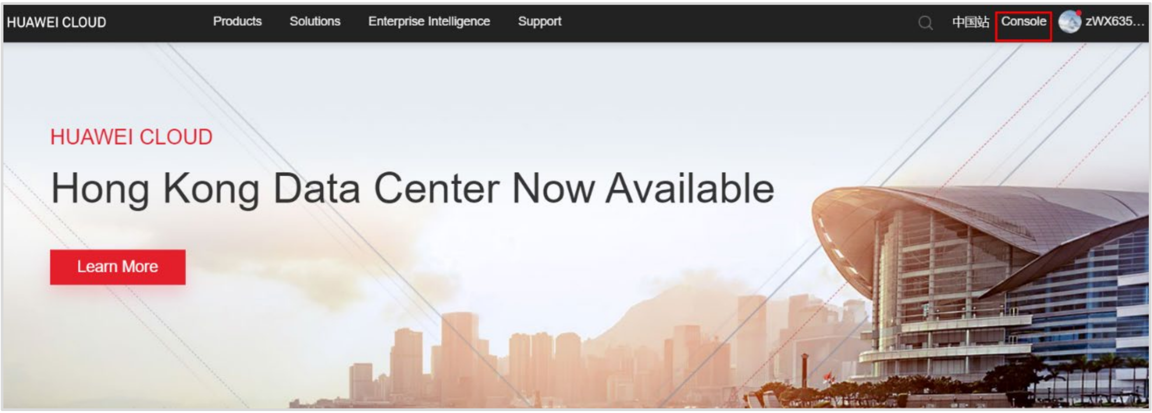


Figure 2-7 HUAWEI CLOUD official website

Step 2 Click My Credential under My Account.

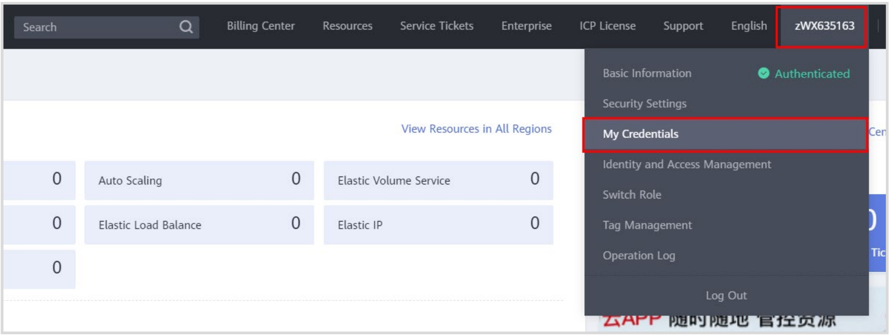
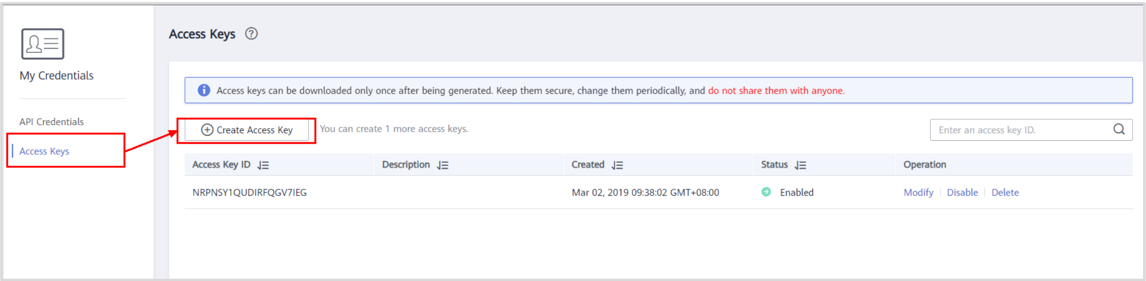


Figure 2-8 consoles

Step 3 Click Access Key to add an access key. After you perform the steps in, the system automatically generates a .csv file. The key is stored in the file. Keep the file secure.



Access keys can be downloaded only once after being generated. Keep them secure, change them periodically, and do not share them with anyone.

You can create 1 more access keys.

| Access Key ID | Description | Created | Status | Operation |
|---|---|---|---|---|
| NRPNSY1QUDIRFQGV7IEG | | Mar 02, 2019 09:38:02 GMT+08:00 | Enabled | Modify \| Disable \| Delete |

Figure 2-9 AK/SK configuration

## 2.4.4 Opening the Jupyter Notebook

You can use the local environment (python 3.6 or 3.7 are recommended) or HUAWEI CLOUD Modelarts Tensorflow 1.8 kernel environment.

Annotation: tensorflow 1.8 kernel environment is not used to only to make sure the code can the code run correctly.

## 2.4.5 Downloading a Dataset

Dataset andSDKis integrated into a compressed file. The link is as follows: https://huawei-ai-certification.obs.cn-north-4.myhuaweicloud.com/ENG/HCIP-AI%20EI%20Developer/V2.1/huaweiei_AIphones.zip

After the download is complete, decompress the package to the related folder.

## 2.4.6 Initialize Image Tag Service

Step 1 Importing Related Libraries

```
# import the package from the image recognition package, image tag, and tool package.
from image_sdk.utils import encode_to_base64
from image_sdk.image_tagging import image_tagging_aksk
from image_sdk.utils import init_global_env

# Invoke JSON to parse the returned result.
import json
# Packages of operating system files or folders
import os
import shutil
# Packages related to image processing and display

from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
```

Step 2 Set related parameters.

```
init_global_env('cn-north-4')

# Prepare AK and SK.
app_key = '*** Change it to your own ak***'
app_secret = '*** Change it to your own sk***'
```

Step 3 Using network image to test

```
# Use the network image test.
demo_data_url = 'https://sdk-obs-source-save.obs.cn-north-4.myhuaweicloud.com/tagging-normal.jpg'
# call interface use the url
result = image_tagging_aksk(app_key, app_secret, '', demo_data_url, 'en', 5, 30)

# Convert the value to a Python dictionary.
tags = json.loads(result)
print(tags)
```

Output:

```
{'result': {'tags': [{'confidence': '98.38', 'i18n_tag': {'en': 'Person', 'zh': '人'}, 'tag': 'Person',
'type': 'object'}, {'confidence': '97.12', 'i18n_tag': {'en': 'Children', 'zh': '儿童'}, 'tag': 'Children',
'type': 'object'}, {'confidence': '96.39', 'i18n_tag': {'en': 'Sandbox', 'zh': '(供儿童玩的)沙坑'},
'tag': 'Sandbox', 'type': 'scene'}, {'confidence': '89.28', 'i18n_tag': {'en': 'Play', 'zh': '玩耍'}, 'tag':
```

```
'Play', 'type': 'object'}, {'confidence': '87.99', 'i18n_tag': {'en': 'Toy', 'zh': '玩具'}, 'tag': 'Toy',
'type': 'object'}]}}
```

## 2.4.7 Labeling related photos

Step 1 Mark a photo

```
# Determine the location of the electronic album.
file_path ='data/'
file_name = 'pic3.jpg'

# Save the image label dictionary.
labels={}

# Image marking
result = image_tagging_aksk(app_key, app_secret, encode_to_base64(file_path + file_name), '','en', 5,
60)
# Parse result.
result_dic = json.loads(result)
# Save the data to the dictionary.
labels[file_name] = result_dic['result']['tags']
print(labels)
```

Output:

```
{'pic3.jpg': [{'confidence': '95.41', 'i18n_tag': {'en': 'Lion', 'zh': '狮子'}, 'tag': 'Lion', 'type':
'object'}, {'confidence': '91.03', 'i18n_tag': {'en': 'Carnivora', 'zh': '食肉目'}, 'tag': 'Carnivora',
'type': 'object'}, {'confidence': '87.23', 'i18n_tag': {'en': 'Cat', 'zh': '猫'}, 'tag': 'Cat', 'type':
'object'}, {'confidence': '86.97', 'i18n_tag': {'en': 'Animal', 'zh': '动物'}, 'tag': 'Animal', 'type':
'object'}, {'confidence': '74.84', 'i18n_tag': {'en': 'Hairy', 'zh': '毛茸茸'}, 'tag': 'Hairy', 'type':
'object'}]}
```

Step 2 Mark all photos in the data folder.

```
# Determine the location of the electronic album.
file_path ='data/'
# Save the image label dictionary.
labels = {}

items = os.listdir(file_path)
for i in items:
    # Check whether the file is a file, not a folder.
    if os.path.isfile:
        # HUAWEI CLOUD EI supports images in JPG, PNG, and BMP formats.
        if i.endswith('jpg') or i.endswith('jpeg') or i.endswith('bmp') or i.endswith('png'):
            # Label images.
            result = image_tagging_aksk(app_key, app_secret, encode_to_base64(file_path + i),
'','en', 5, 60)
            # Parse the returned result.
            result_dic = json.loads(result)
            # Align the file name with the image.
            labels[i] = result_dic['result']['tags']
```

```
# Display the result.
print(labels)
```

Output:

```
{'pic1.jpg': [{'confidence': '89.73', 'i18n_tag': {'en': 'Running', 'zh': '奔跑'}, 'tag': 'Running',
'type': 'object'}, {'confidence': '88.34', 'i18n_tag': {'en': 'Person', 'zh': '人'}, 'tag': 'Person', 'type':
'object'}, {'confidence': '87.59', 'i18n_tag': {'en': 'Motion', 'zh': '运动'}, 'tag': 'Motion', 'type':
'object'}, {'confidence': '87.24', 'i18n_tag': {'en': 'Sunrise', 'zh': '日出'}, 'tag': 'Sunrise', 'type':
'object'}, {'confidence': '86.68', 'i18n_tag': {'en': 'Outdoors', 'zh': '户外'}, 'tag': 'Outdoors',
'type': 'object'}], 'pic10.jpg': [{'confidence': '85.83', 'i18n_tag': {'en': 'Flower', 'zh': '花朵'}, 'tag':
'Flower', 'type': 'object'}, {'confidence': '84.33', 'i18n_tag': {'en': 'Plant', 'zh': '植物'}, 'tag':
'Plant', 'type': 'object'}, {'confidence': '83.47', 'i18n_tag': {'en': 'Red', 'zh': '红色'}, 'tag': 'Red',
'type': 'object'}, {'confidence': '79.92', 'i18n_tag': {'en': 'Flower', 'zh': '花'}, 'tag': 'Flower', 'type':
'object'}, {'confidence': '78.67', 'i18n_tag': {'en': 'Flowers and plants', 'zh': '花卉'}, 'tag':
'Flowers and plants', 'type': 'object'}], 'pic2.jpg': [{'confidence': '99.61', 'i18n_tag': {'en': 'Cat',
'zh': '猫'}, 'tag': 'Cat', 'type': 'object'}, {'confidence': '99.22', 'i18n_tag': {'en': 'Carnivora', 'zh':
'食肉目'}, 'tag': 'Carnivora', 'type': 'object'}, {'confidence': '88.96', 'i18n_tag': {'en': 'Field road',
'zh': '田野路'}, 'tag': 'Field road', 'type': 'scene'}, {'confidence': '86.12', 'i18n_tag': {'en':
'Animal', 'zh': '动物'}, 'tag': 'Animal', 'type': 'object'}, {'confidence': '83.33', 'i18n_tag': {'en':
'Mammal', 'zh': '哺乳动物'}, 'tag': 'Mammal', 'type': 'object'}], 'pic3.jpg': [{'confidence':
'95.41', 'i18n_tag': {'en': 'Lion', 'zh': '狮子'}, 'tag': 'Lion', 'type': 'object'}, {'confidence': '91.03',
'i18n_tag': {'en': 'Carnivora', 'zh': '食肉目'}, 'tag': 'Carnivora', 'type': 'object'}, {'confidence':
'87.23', 'i18n_tag': {'en': 'Cat', 'zh': '猫'}, 'tag': 'Cat', 'type': 'object'}, {'confidence': '86.97',
'i18n_tag': {'en': 'Animal', 'zh': '动物'}, 'tag': 'Animal', 'type': 'object'}, {'confidence': '74.84',
'i18n_tag': {'en': 'Hairy', 'zh': '毛茸茸'}, 'tag': 'Hairy', 'type': 'object'}], 'pic4.jpg': [{'confidence':
'92.35', 'i18n_tag': {'en': 'Retro', 'zh': '复古'}, 'tag': 'Retro', 'type': 'object'}, {'confidence':
'91.39', 'i18n_tag': {'en': 'Design', 'zh': '设计'}, 'tag': 'Design', 'type': 'object'}, {'confidence':
'86.89', 'i18n_tag': {'en': 'Home furnishing', 'zh': '家居'}, 'tag': 'Home furnishing', 'type':
'object'}, {'confidence': '86.43', 'i18n_tag': {'en': 'Bow window indoor', 'zh': '弓形窗/室内'}...
(omit)
```

Step 3 Save the marking result.

```
# Save the label dictionary to a file.
save_path = './label'
# If the folder does not exist, create a file.
if not os.path.exists(save_path):
    os.mkdir(save_path)

# Create a file, write the file, and close the file.
with open(save_path + '/labels.json', 'w+') as f:
    f.write(json.dumps(labels))
```

## 2.4.8 Making Dynamic Album by Using Marking Results

Step 1 Reopen the saved labeling result.

```
# Open the saved file.
label_path = 'label/labels.json'
with open(label_path, 'r') as f:
    labels = json.load(f)
```

Step 2 Use keywords to search (the keyword is Flower).

```
# Search keyword
key_word = input('Please enter a keyword.')

# Set the trusted percentage.
threshold = 60
# Set a collection (the collection contains only one element).
valid_list = set()

# Traverse the dictionary in labels to obtain all image names that contain keywords.
for k,v in labels.items():
    for item in v:
        if key_word in item['tag'] and float(item['confidence']) >= threshold:
            valid_list.add(k)




# Display the result.
valid_list = list(valid_list)
print(valid_list)
```

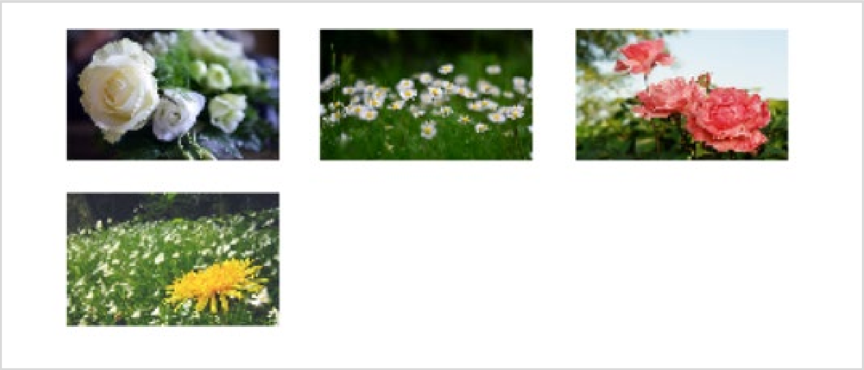
Output:

```
Please enter a keyword.
['pic10.jpg', 'pic7.jpg', 'pic5.jpg', 'pic9.jpg']
```

Step 3 Display related images.

```
# Set the canvas size.
plt.figure(24)

# Arrange each image on the canvas in sequence.
for k,v in enumerate(valid_list[:9]):
    pic_path = 'data/' + v
    img = Image.open(pic_path)
    img = img.resize((640, 400))
    plt.subplot(331 + k)
    plt.axis('off')
    plt.imshow(img)

plt.show()
```

Output:



Step 4 Creating a GIF Image

```
# Generate a temporary folder.
if not os.path.exists('tmp'):
    os.mkdir('tmp')

# Convert all searched images into GIF format and store them in a temporary folder.
gif_list = []
for k, pic in enumerate(valid_list):
    pic_path = 'data/' + pic
    img = Image.open(pic_path)
    img = img.resize((640, 380))
    save_name = 'tmp/'+ str(k) + '.gif'
    img.save(save_name)
    gif_list.append(save_name)

# Open all static GIF images.
images=[]
for i in gif_list:
    pic_path = i
    images.append(Image.open(pic_path))

# Save the GIF image.
images[0].save('Album Animation.gif',
                save_all=True,
                append_images=images[1:],
                duration=1000,
                loop=0)

# Release the memory.
del images
# Delete the temporary folder.
shutil.rmtree('tmp')

print('GIF album created.')
```

Output:

GIF album created.

### 2.4.9 Automatically classify photos with labels

Step 1 Automatic classification

```
# Open the saved labels file.
label_path = 'label/labels.json'
with open(label_path, 'r') as f:
    labels = json.load(f)

# Obtain the file category with the highest confidence.
classes =[[v[0]['tag'] ,k] for k, v in labels.items()]

for cls in classes:
    if not os.path.exists('data/' + cls[0]):
        os.mkdir('data/'+ cls[0])
    # Copy the corresponding image.
    shutil.copy('data/'+ cls[1], 'data/'+ cls[0]+ '/'+ cls[1])

print('Copying completed.')
```

Output:

```
Copying completed
```

## 2.5 Experiment Summary

This experiment describes how to use Image Tag service to perform operations related to electronic albums. First, this experiment describes how to enable services under image recognition. Second the experiment focuses on how to use Image Tag to label photos, search for albums, and create dynamic albums, automatically classify photos and display related results. In addition, we have practiced and performed basic operations on the image recognition libraries of HUAWEI CLOUD EI service.

Huawei AI Certification Training

# HCIP-AI-EI Developer

# Speech Processing Lab Guide

ISSUE:2.0

HUAWEI TECHNOLOGIES CO., LTD.

**Trademarks and Permissions**

HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

**Notice**

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

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

**Huawei Technologies Co., Ltd.**

Address: Huawei Industrial Base Bantian, Longgang Shenzhen 518129
People's Republic of China

Website: http://e.huawei.com

# Huawei Certificate System

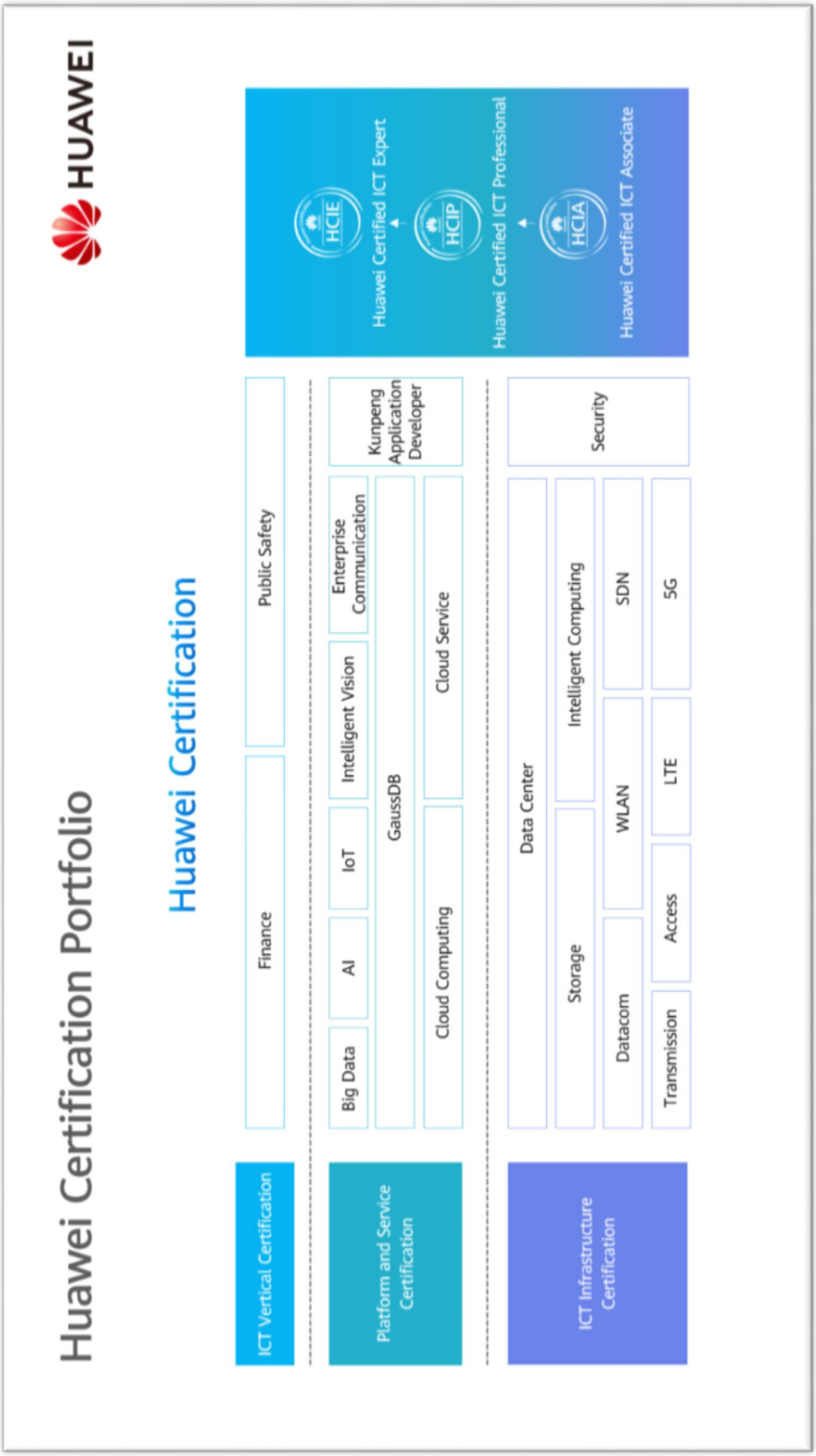
Huawei's certification system is the industry's only one that covers all ICT technical fields. It is developed relying on Huawei's 'platform + ecosystem' strategy and new ICT technical architecture featuring cloud-pipe-device synergy. It provides three types of certifications: ICT Infrastructure Certification, Platform and Service Certification, and ICT Vertical Certification.

To meet ICT professionals' progressive requirements, Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

HCIP-AI-EI Developer V2.0 certification is intended to cultivate professionals who have acquired basic theoretical knowledge about image processing, speech processing, and natural language processing and who are able to conduct development and innovation using Huawei enterprise AI solutions (such as HUAWEI CLOUD EI), general open-source frameworks, and ModelArts, a one-stop development platform for AI developers.

The content of HCIP-AI-EI Developer V2.0 certification includes but is not limited to: neural network basics, image processing theory and applications, speech processing theory and applications, natural language processing theory and applications, ModelArts overview, and image processing, speech processing, natural language processing, and ModelArts platform development experiments. ModelArts is a one-stop development platform for AI developers. With data preprocessing, semi-automatic data labeling, large-scale distributed training, automatic modeling, and on-demand model deployment on devices, edges, and clouds, ModelArts helps AI developers build models quickly and manage the lifecycle of AI development. Compared with V1.0, HCIP-AI-EI Developer V2.0 adds the ModelArts overview and development experiments. In addition, some new EI cloud services are updated.

HCIP-AI-EI Developer V2.0 certification proves that you have systematically understood and mastered neural network basics, image processing theory and applications, speech processing theory and applications, ModelArts overview, natural language processing theory and applications, image processing application development, speech processing application development, natural language processing application development, and ModelArts platform development. With this certification, you will acquire (1) the knowledge and skills for AI pre-sales technical support, AI after-sales technical support, AI product sales, and AI project management; (2) the ability to serve as an image processing developer, speech processing developer, or natural language processing developer.



Huawei Certification Portfolio

HUAWEI

Huawei Certification

ICT Vertical Certification

Finance

Public Safety

Platform and Service Certification

Big Data

AI

IoT

Intelligent Vision

Enterprise Communication

Kunpeng Application Developer

GaussDB

Cloud Computing

Cloud Service

ICT Infrastructure Certification

Data Center

Security

Storage

Intelligent Computing

Datacom

WLAN

SDN

Transmission

Access

LTE

5G

HCIE

Huawei Certified ICT Expert

HCIP

Huawei Certified ICT Professional

HCIA

Huawei Certified ICT Associate

# About This Document

## Overview

This document is an HCIP-AI certification training course. It is intended for trainees who are preparing for HCIP-AI tests or readers who want to know about AI basics. After understanding this document, you will be able to perform speech processing, for example, speech file pre-processing, speech input, text to speech (TTS), and automatic speech recognition (ASR), and carry out development. To implement the ASR operations, we use the TensorFlow framework to construct the deep neural network, such as Seq2Seq model.

## Description

This document contains three experiments and it involves speech file pre-processing, Huawei-based TTS and ASR. It aims to improve the practical development capability of AI speech processing.

- Experiment 1: helps understand Python-based speech file pre-processing.
- Experiment 2: helps understand how to implement TTS through HUAWEI CLOUD EI.
- Experiment 3 helps understand Tensorflow-based ASR.

## Background Knowledge Required

- Have basic Python language programming skills.
- Have basic knowledge in speech processing.
- Have basic knowledge in TensorFlow and Keras.
- Have basic knowledge in deep neural network.

## Experiment Environment Overview

- Windows (64-bit)
- Anaconda3 (64-bit) (Python 3.6.4 or later)
- Jupyter Notebook
- Link for downloading the experiment data:

https://huawei-ai-certification.obs.cn-north-4.myhuaweicloud.com/ENG/HCIP-AI%20EI%20Developer/V2.1/speech.rar

- Speech Pre-processing.
- TTS based on HUAWEI CLOUD EI.
- ASR based on Seq2Seq

# Contents

# 1 Speech Preprocessing

## 1.1 Introduction

### 1.1.1 About this lab

Speech is a non-stationary time-varying signal. It carries various information. Information including in speech needs to be extracted for speech processing, for example, speech encoding, TTS, speech recognition, and speech quality enhancement. Generally, speech data is processed to analyze speech signals and extract characteristic parameters for subsequent processing or to process speech signals. For example, background noise is suppressed in speech quality enhancement to obtain relatively "clean" speech. In TTS, splicing and smoothing need to be performed for speech segments to obtain synthetic speech with higher subjective speech quality. Applications in this aspect are also created on the basis of analysis and extraction of speech signal information. In a word, the purpose of speech signal analysis is to conveniently and effectively extract and express information carried in speech signals.

Based on types of analyzed parameters, speech signal analysis can be divided into time-domain analysis and transform-domain (frequency domain and cepstral domain) analysis. The time-domain analysis method is the simplest and the most intuitive method. It directly analyzes time-domain waveforms of speech signals and extracts characteristic parameters, including short-time energy and average amplitude of speech, average short-time zero-crossing rate, short-time autocorrelation function, and short-time average amplitude difference function.

This experiment provides analysis based on speech data attributes of the short-sequence speech data set and related characteristic attributes to have a more in-depth and comprehensive understanding of speech data.

### 1.1.2 Objectives

Upon completion of this task, you will be able to:

- Check the attributes of speech data.
- Understand the features of speech data.

### 1.1.3 Knowledge Required

This experiment requires knowledge in two aspects:

- Syntactical basis of the Python language and hands-on operation capability.
- Understanding of the related wave framework.

## 1.2 Installing Related Modules

Install the Python module.

Click Start in the lower left corner of the Windows OS. A menu list is displayed.

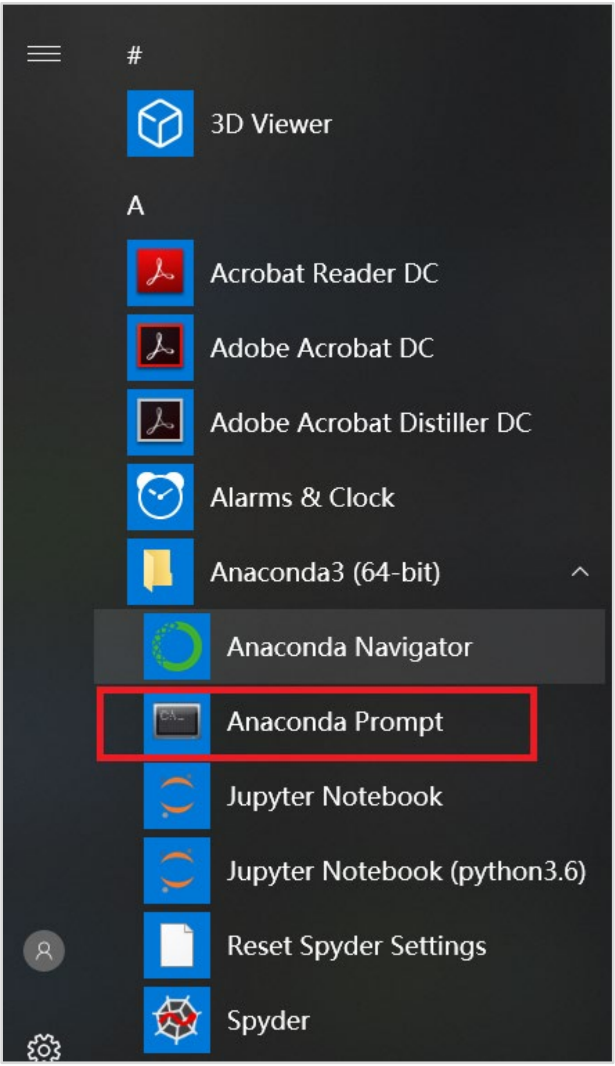


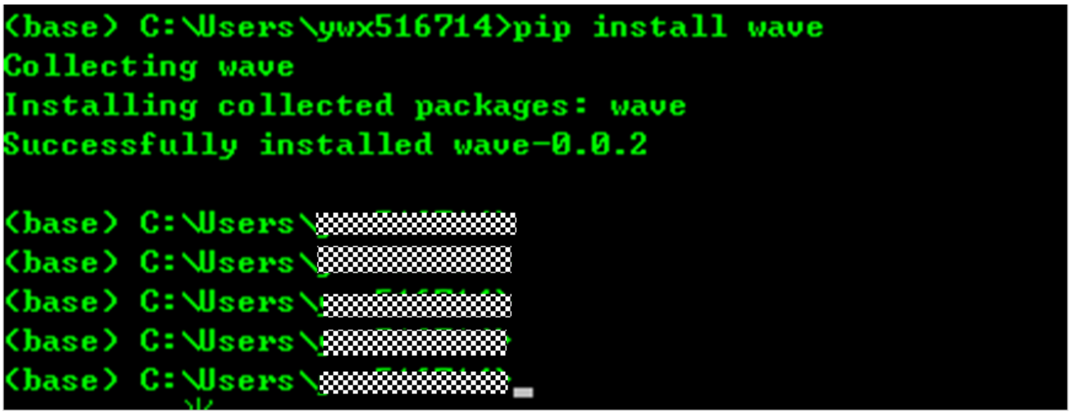
**Figure 1-1 Anaconda Prompt**

Click Anaconda Prompt. The Anaconda system is displayed.

```
(base) C:\Users\y[redacted]>
```

**Figure 1-2 Anaconda Prompt**

Install the wave module. Enter pip install wave. The result is as follows:



```
(base) C:\Users\ywx516714>pip install wave
Collecting wave
Installing collected packages: wave
Successfully installed wave-0.0.2

(base) C:\Users\[redacted]
(base) C:\Users\[redacted]
(base) C:\Users\[redacted]
(base) C:\Users\[redacted]
(base) C:\Users\[redacted]
```

**Figure 1-3 Install wave**

Install other required Python frameworks by following the similar steps.

## 1.3 Procedure

This experiment is performed based on the wave framework. Main steps include:

- View audio data attributes.
- View audio data conversion matrix
- View the audio spectrum.
- View the audio waveform.

Step 1 Import related modules

Code:

```
import wave as we
import matplotlib.pyplot as plt
import numpy as np
from scipy.io import wavfile
import matplotlib.pyplot as plt
from matplotlib.backend_bases import RendererBase
from scipy import signal
from scipy.io import wavfile
import os
from scipy.fftpack import fft
import warnings
warnings.filterwarnings("ignore")
```

## Step 2 View basic attributes of the wav file

Code:

```
filename = 'data/thchs30/train/A2_0.wav '
WAVE = we.open(filename)
# Output information (sound channel, sampling width, frame rate, number of frames, unique ID, and
# lossless information)
for item in enumerate(WAVE.getparams()):
      print (item)
a = WAVE.getparams().nframes     # Total number of frames
print(a)
f = WAVE.getparams().framerate   # Sampling frequency
print("Sampling frequency：",f)
sample_time = 1/f                          # Interval of sampling points
time = a/f                                 # Sound signal length
sample_frequency, audio_sequence = wavfile.read(filename)
print (audio_sequence,len(audio_sequence ))
x_seq = np.arange(0,time,sample_time)
print(x_seq,len(x_seq))
```

Result:

```
(0, 1)
(1, 2)
(2, 16000)
(3, 157000)
(4, 'NONE')
(5, 'not compressed')
157000
Sampling frequency：  16000
[-296 -424 -392 ... -394 -379 -390] 157000
[0.0000000e+00 6.2500000e-05 1.2500000e-04 ... 9.8123125e+00 9.8123750e+00
 9.8124375e+00] 157000
```

## Step 3 View the waveform sequence of the wav file

Code:

```
plt.plot(x_seq,audio_sequence, 'blue' )
plt.xlabel('time (s)')
plt.show()
```

Result:

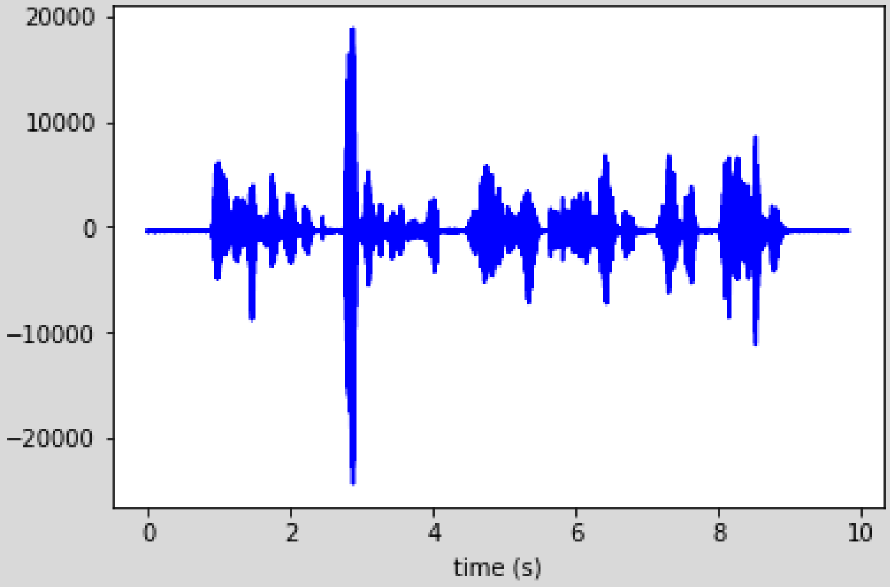


**Figure 1-4 The waveform sequence of the wav file**

Step 4 Obtain file name

Code:

```
audio_path = 'data/train/audio/'
pict_Path = 'data/train/audio/'
samples = []
# Verify that the file exists, if not here, create it
if not os.path.exists(pict_Path):
    os.makedirs(pict_Path)

subFolderList = []
for x in os.listdir(audio_path):
    if os.path.isdir(audio_path + '/' + x):
        subFolderList.append(x)
        if not os.path.exists(pict_Path + '/' + x):
            os.makedirs(pict_Path +'/'+ x)
# View the name and number of sub-files
print("----list----:",subFolderList)
print("----len----:",len(subFolderList))
```

Result:

```
----list----: ['bed', 'bird', 'cat', 'dog', 'down', 'eight', 'five', 'four', 'go', 'happy', 'house', 'left', 'marvin',
'nine', 'no', 'off', 'on', 'one', 'right', 'seven', 'sheila', 'six', 'stop', 'three', 'tree', 'two', 'up', 'wow', 'yes',
'zero', '_background_noise_']
----len----: 31
```

### Step 5 Count the number of speech files in each subfolder

Code:

```
sample_audio = []
total = 0
for x in subFolderList:
    # Get all wav files
    all_files = [y for y in os.listdir(audio_path + x) if '.wav' in y]
    total += len(all_files)
    sample_audio.append(audio_path   + x + '/'+ all_files[0])
    # View the number of files in each subfolder
    print('%s : count: %d ' % (x , len(all_files)))
# View the total number of wav files
print("TOTAL:",total)
```

Result:

```
bed : count: 10
bird : count: 15
cat : count: 17
dog : count: 20
down : count: 36
eight : count: 16
five : count: 16
four : count: 22
go : count: 18
happy : count: 16
house : count: 15
left : count: 20
marvin : count: 19
nine : count: 14
no : count: 16
off : count: 20
on : count: 11
one : count: 18
right : count: 22
seven : count: 20
sheila : count: 17
six : count: 15
stop : count: 12
three : count: 19
tree : count: 14
two : count: 12
up : count: 10
wow : count: 18
yes : count: 17
zero : count: 20
_background_noise_ : count: 6
TOTAL: 521
```

### Step 6 View the first file in each sub-folder

Code:

```
for x in sample_audio:
    print(x)
```

**Result:**

```
data/train/audio//bed/00f0204f_nohash_0.wav
data/train/audio//bird/00b01445_nohash_0.wav
data/train/audio//cat/00b01445_nohash_0.wav
data/train/audio//dog/fc2411fe_nohash_0.wav
data/train/audio//down/fbdc07bb_nohash_0.wav
data/train/audio//eight/fd395b74_nohash_0.wav
data/train/audio//five/fd395b74_nohash_2.wav
data/train/audio//four/fd32732a_nohash_0.wav
data/train/audio//go/00b01445_nohash_0.wav
data/train/audio//happy/fbf3dd31_nohash_0.wav
data/train/audio//house/fcb25a78_nohash_0.wav
data/train/audio//left/00b01445_nohash_0.wav
data/train/audio//marvin/fc2411fe_nohash_0.wav
data/train/audio//nine/00b01445_nohash_0.wav
data/train/audio//no/fe1916ba_nohash_0.wav
data/train/audio//off/00b01445_nohash_0.wav
data/train/audio//on/00b01445_nohash_0.wav
data/train/audio//one/00f0204f_nohash_0.wav
data/train/audio//right/00b01445_nohash_0.wav
data/train/audio//seven/0a0b46ae_nohash_0.wav
data/train/audio//sheila/00f0204f_nohash_0.wav
data/train/audio//six/00b01445_nohash_0.wav
data/train/audio//stop/0ab3b47d_nohash_0.wav
data/train/audio//three/00b01445_nohash_0.wav
data/train/audio//tree/00b01445_nohash_0.wav
data/train/audio//two/00b01445_nohash_0.wav
data/train/audio//up/00b01445_nohash_0.wav
data/train/audio//wow/00f0204f_nohash_0.wav
data/train/audio//yes/00f0204f_nohash_0.wav
data/train/audio//zero/0ab3b47d_nohash_0.wav
data/train/audio//_background_noise_/doing_the_dishes.wav
```

Step 7 Create a spectrum processing function

Code:

```
def log_specgram(audio, sample_rate, window_size=20,
                 step_size=10, eps=1e-10):
    nperseg = int(round(window_size * sample_rate / 1e3))
    noverlap = int(round(step_size * sample_rate / 1e3))
    freqs, _, spec = signal.spectrogram(audio,
                                        fs=sample_rate,
                                        window='hann',
                                        nperseg=nperseg,
                                        noverlap=noverlap,
                                        detrend=False)
    return freqs, np.log(spec.T.astype(np.float32) + eps)
```

Step 8 Visualize one spectrum of multiple samples

Code:

```
fig = plt.figure(figsize=(20,20))

for i, filepath in enumerate(sample_audio[:16]):
    # Make subplots
    plt.subplot(4,4,i+1)

    # pull the labels
    label = filepath.split('/')[-2]
    plt.title(label)

    # create spectrogram
    samplerate, test_sound   = wavfile.read(filepath)
    _, spectrogram = log_specgram(test_sound, samplerate)

    plt.imshow(spectrogram.T, aspect='auto', origin='lower')
    plt.axis('off')
plt.show()
```

Result:

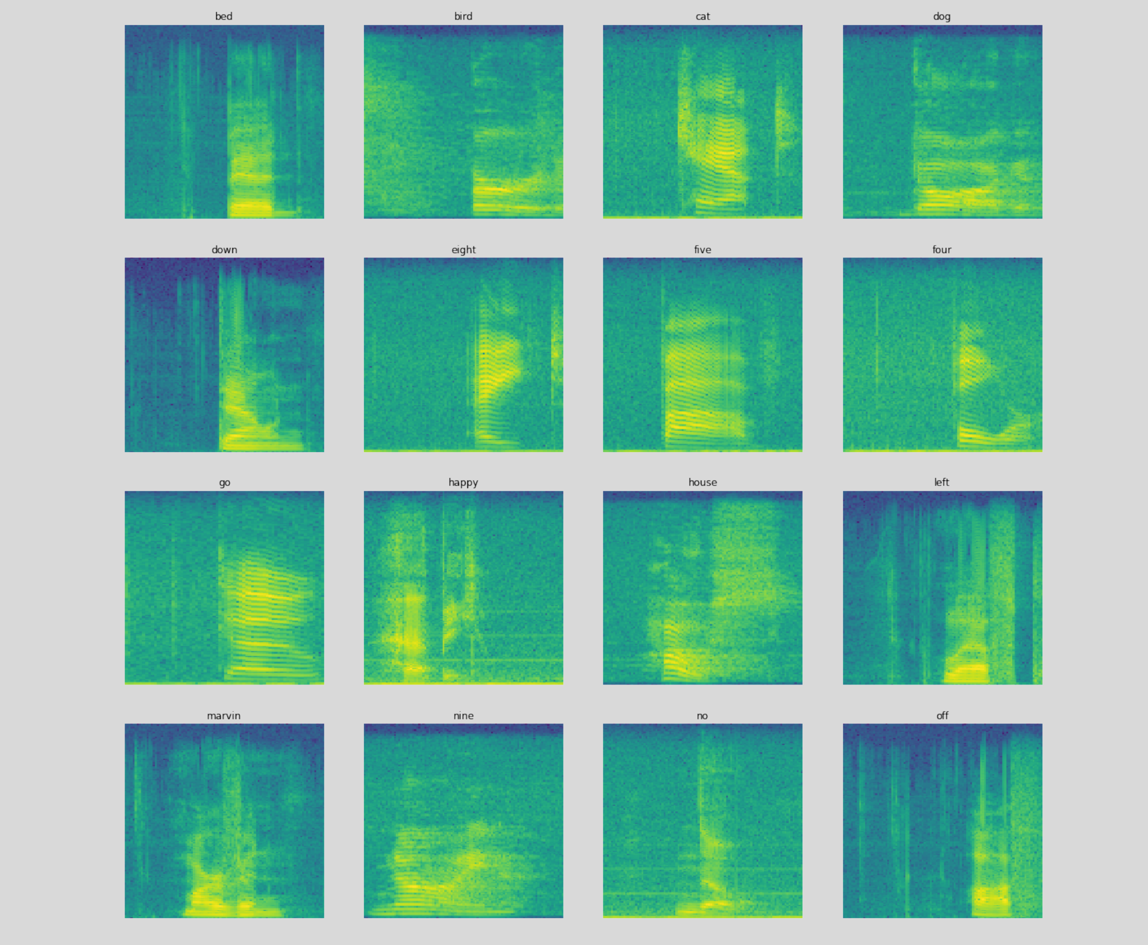


**Figure 1-5 One spectrum of multiple samples**

Step 9 Visualize multiple spectrums of one sample

Code:

```
yes_samples = [audio_path + 'yes/' + y for y in os.listdir(audio_path + 'yes/')[:9]]
fig = plt.figure(figsize=(10,10))

for i, filepath in enumerate(yes_samples):
    # Make subplots
    plt.subplot(3,3,i+1)

    # pull the labels
    label = filepath.split('/')[-1]
    plt.title('"yes": '+label)

    # create spectrogram
    samplerate, test_sound    = wavfile.read(filepath)
    _, spectrogram = log_specgram(test_sound, samplerate)

    plt.imshow(spectrogram.T, aspect='auto', origin='lower')
    plt.axis('off')
plt.show()
```

Result:

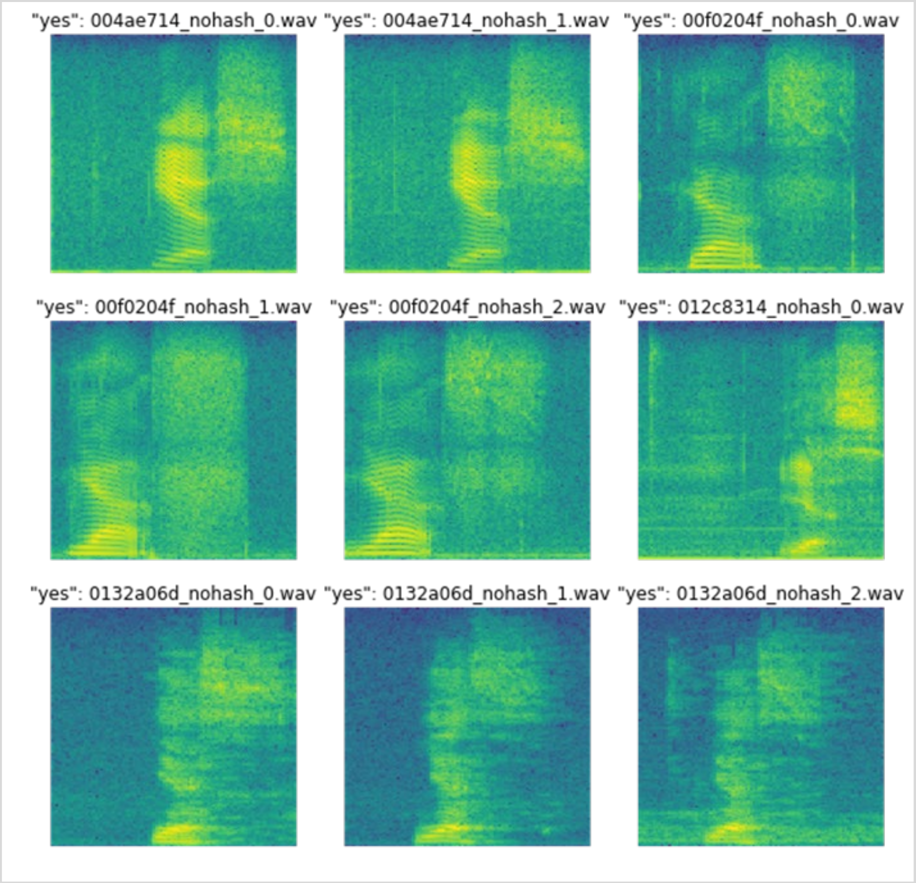


**Figure 1-6 Multiple spectrums of one sample**

Step 10 Visualize the waveforms of multiple samples

Code:

```
fig = plt.figure(figsize=(10,10))
for i, filepath in enumerate(sample_audio[:16]):
    plt.subplot(4,4,i+1)
    samplerate, test_sound   = wavfile.read(filepath)
    plt.title(filepath.split('/')[-2])
    plt.axis('off')
    plt.plot(test_sound)
plt.show()
```

Result:

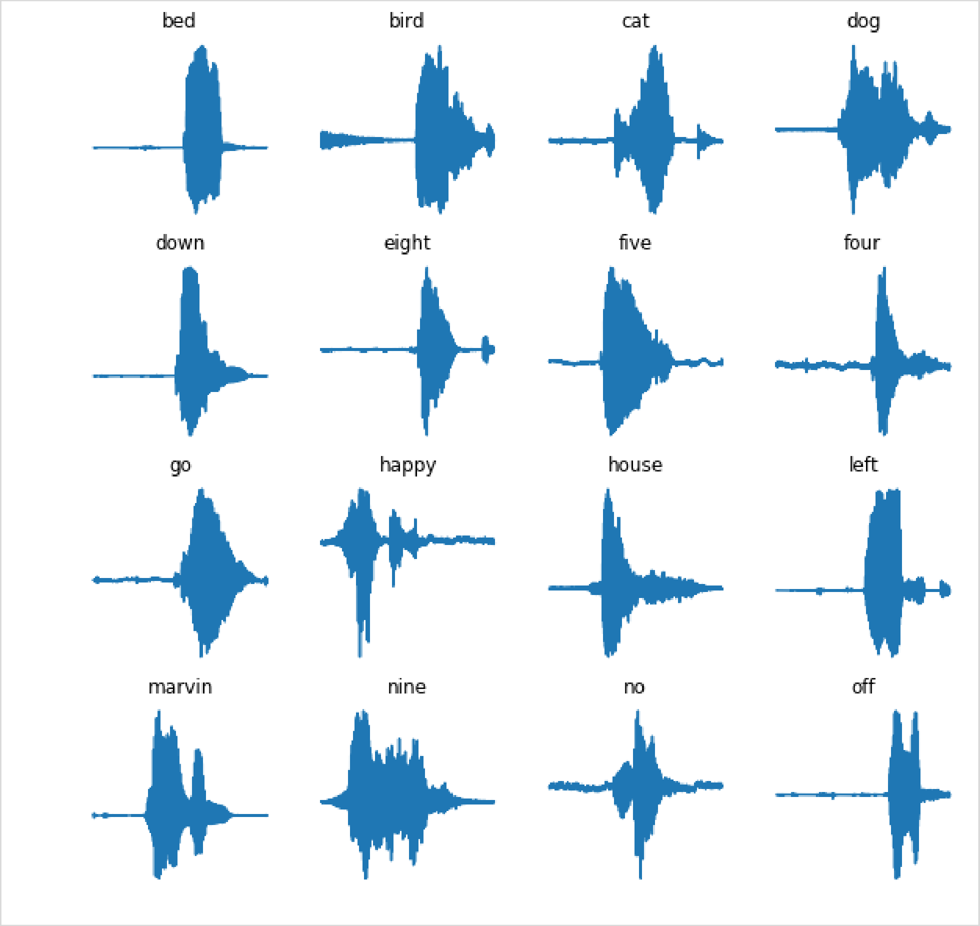


**Figure 1-7 The waveforms of multiple samples**

Step 11 Visualize multiple waveforms of one sample

Code:

```
fig = plt.figure(figsize=(8,8))
```

```
for i, filepath in enumerate(yes_samples):
    plt.subplot(3,3,i+1)
    samplerate, test_sound = wavfile.read(filepath)
    plt.title(filepath.split('/')[-2])
    plt.axis('off')
    plt.plot(test_sound)
plt.show()
```

Result:

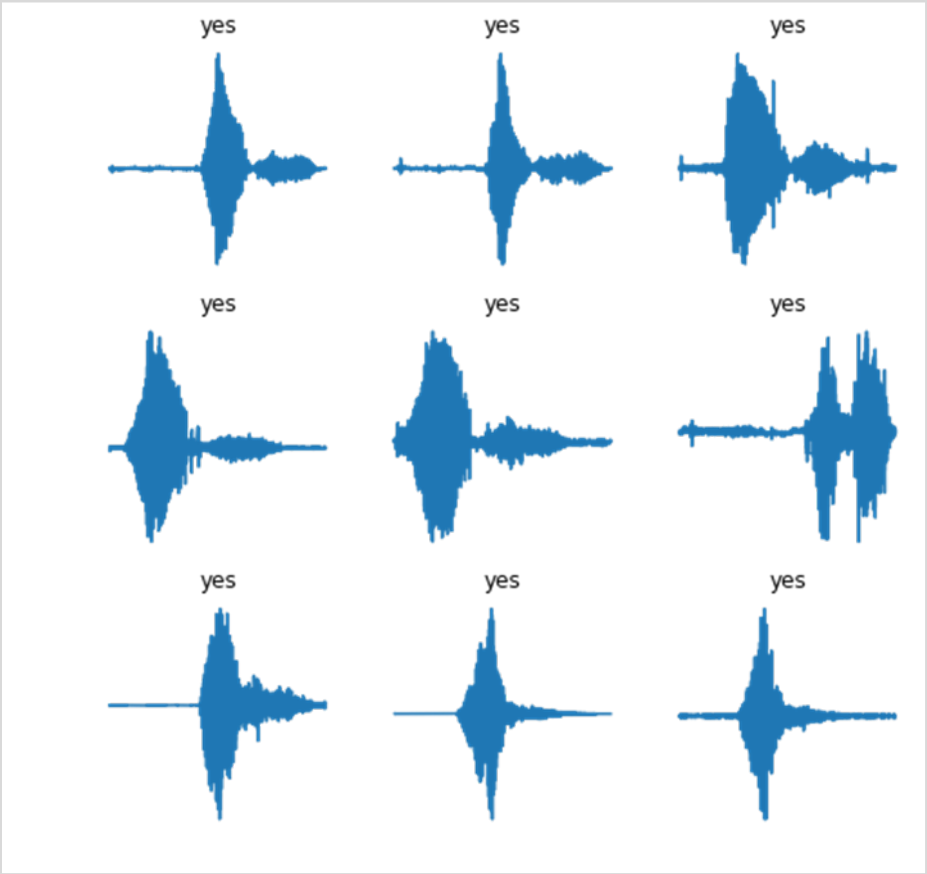


**Figure 1-8 Multiple waveforms of one sample**

## 1.4 Summary

This experiment is a speech data pre-processing experiment based on the Python language, wave speech processing framework, and open source data set. It mainly includes viewing of basic speech data and processing of waveform and spectrum files. Visualization and display of specific values help trainees view essential attributes of speech data more clearly.

# 2 HUAWEI CLOUD EI Text-to-Speech Service

## 2.1 Introduction

### 2.1.1 About this lab

In the Speech Interaction Service on Huawei Cloud, there are text to speech and speech recognition services. The content of this experiment is a customized version of text to speech and a customized version of a single sentence recognition service.

Text To Speech (TTS), is a service that converts texts into realistic voices. TTS provides users with open application programming interfaces (APIs). Users can obtain the TTS result by accessing and calling APIs in real time and synthesize the input text into audio. Personalized voice services are provided for enterprises and individuals by selecting tone, customizing the volume and speed.

This service can release the Restful HTTP request service of the POST in either of the following ways: by calling the underlying interface encapsulated by the SDK to release the Restful service, or by simulating the access of the frontend browser. The former requires the AK and SK of the user for identity authentication. The latter requires the user token for identity authentication. In this lab, AK/SK authentication is used to publish a request service.

### 2.1.2 Objectives

Upon completion of this task, you will be able to:

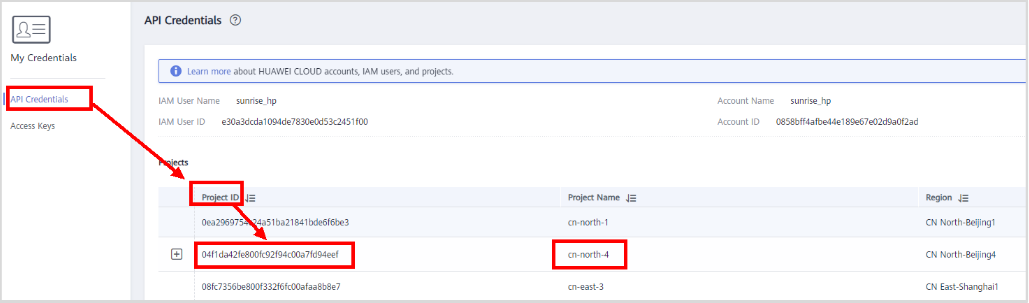
- Learn how to use HUAWEI CLOUD to perform text to speech and speech recognition.
- Understand and master how to use Python to develop services.

## 2.2 Preparing the Experiment Environment

- Registering and Logging In to the HUAWEI CLOUD Management Console.
- For details about the documents related to speech synthesis and speech recognition, see https://support.huaweicloud.com/en-us/api-sis/sis_03_0111.html and https://support.huaweicloud.com/en-us/api-sis/sis_03_0040.html.
- Prepare the AK/SK of the HUAWEI CLOUD account. If you can get it before, you can continue to use the previous AK/SK. If you have not obtained AK/SK before, you can log in HUAWEI CLOUD, click "My Credentials" in the user name, and select **Access**

**Keys**> **Create Access Key** on the "My Credentials" interface to obtain and download. Please keep the AK/SK information properly. You do not need to add any more in other experiments, you can use this AK/SK directly.

- Prepare project_id. If you have obtained it before, you can continue to use the previous project ID. If you have not obtained it, you can view the project ID in the API Credentials on the "My Credentials" interface, and copy the project ID of the region as your project_id.



API Credentials

Learn more about HUAWEI CLOUD accounts, IAM users, and projects.

IAM User Name sunrise_hp
IAM User ID e30a3dcda1094de7830e0d53c2451f00
Account Name sunrise_hp
Account ID 0858bff4afbe44e189e67e02d9a0f2ad

Projects

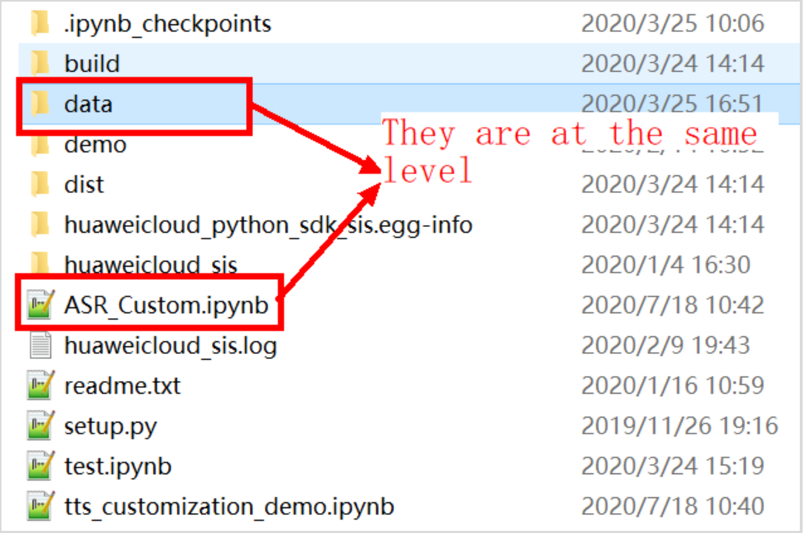
| Project ID | Project Name | Region |
|---|---|---|
| 0ea2969754[illegible]24a51ba21841bde6f6be3 | cn-north-1 | CN North-Beijing1 |
| 04f1da42fe800fc92f94c00a7fd94eef | cn-north-4 | CN North-Beijing4 |
| 08fc7356be800f332f6fc00afaa8b8e7 | cn-east-3 | CN East-Shanghai1 |

**Figure 2-1 Project ID**

- You need to confirm that the Python environment has been installed, the Python SDK is suitable for Python3, and Python 3.6 or 3.7 is recommended.

## 2.3 Obtaining and configuring Python SDK

1. Download the Python SDK for the Speech Interaction service (https://mirrors.huaweicloud.com/sis-sdk/python/huaweicloud-python-sdk-sis-1.0.0.rar ) and decompress it. We can use the data in the data folder. The code can be at the same level as the data folder. We can also use our own data and put it in the data folder. What is the same level? As shown in the figure below, the files are of the same level.



| | |
|---|---|
| .ipynb_checkpoints | 2020/3/25 10:06 |
| build | 2020/3/24 14:14 |
| data | 2020/3/25 16:51 |
| demo | [illegible] |
| dist | 2020/3/24 14:14 |
| huaweicloud_python_sdk_sis.egg-info | 2020/3/24 14:14 |
| huaweicloud_sis | 2020/1/4 16:30 |
| ASR_Custom.ipynb | 2020/7/18 10:42 |
| huaweicloud_sis.log | 2020/2/9 19:43 |
| readme.txt | 2020/1/16 10:59 |
| setup.py | 2019/11/26 19:16 |
| test.ipynb | 2020/3/24 15:19 |
| tts_customization_demo.ipynb | 2020/7/18 10:40 |

They are at the same level

**Figure 2-2 The Same Level**

2. Please confirm that the Python package management tool “setuptools” has been installed. Please confirm that requests and websocket-client packages have been installed. The installed list can be viewed through the "point list" command. If they are not installed, use the following command to install:

```
pip install setuptools
pip install requests
pip install websocket-client
```

3. Use the Anaconda Prompt command to switch to the Python SDK decompression directory.
4. In the SDK directory, execute the command “python setup.py install” to install the Python SDK to the development environment, or import the .py file directly into the project.

## 2.4 Procedure

This experiment needs to download the SDK of the speech interaction service on the Huawei public cloud service, and use the AK\SK information for identity authentication to call the SDK underlying interface service to submit the Restful service request. This experiment uses the SDK to call the TTS services , And run the experiment in Jupyter Notebook. Specific steps are as follows:

### 2.4.1 TTS

Customized TTS is a service that converts text into realistic speech. The user obtains TTS result by accessing and calling API in real time, and convert the text input by the user into speech. Provide personalized pronunciation services for enterprises and individuals through tone selection, custom volume, and speech speed.

Step 1 Import related modules

Code:

```
# -*- coding: utf-8 -*-
from huaweicloud_sis.client.tts_client import TtsCustomizationClient
from huaweicloud_sis.bean.tts_request import TtsCustomRequest
from huaweicloud_sis.bean.sis_config import SisConfig
from huaweicloud_sis.exception.exceptions import ClientException
from huaweicloud_sis.exception.exceptions import ServerException
import json
```

Step 2 Configure related parameters

Code:

```
ak = "***" #Configure your own ak
sk = "***" #Configure your own sk
```

```
project_id = "***" #Configure your own project_id
region = "cn-north-4" #Beijing-4 is used by default, and the corresponding region code is cn-north-4
```

### Step 3 Configure data and save path

Code:

```
text ='I like you, do you like me?' # The text to be synthesized, no more than 500 words
path ='data/test.wav' #configure save path, you can also choose not to save in the settings
```

### Step 4 Initialize the client

Code:

```
config = SisConfig()
config.set_connect_timeout(5)          # Set connection timeout
config.set_read_timeout(10)            # Set read timeout
ttsc_client = TtsCustomizationClient(ak, sk, region, project_id, sis_config=config)
```

### Step 5 Construct request

Code：

```
ttsc_request = TtsCustomRequest(text)
# Set request, all parameters can be left unset, use default parameters
# Set audio format, default wav, optional mp3 and pcm
ttsc_request.set_audio_format('wav')
#Set the sampling rate, 8000 or 16000, the default is 8000
ttsc_request.set_sample_rate('8000')
# Set the volume, [0, 100], default 50
ttsc_request.set_volume(50)
# Set the pitch, [-500, 500], default 0
ttsc_request.set_pitch(0)
# Set the speed of sound, [-500, 500], default 0
ttsc_request.set_speed(0)
# Set whether to save, the default is False
ttsc_request.set_saved(True)
# Set the save path, this parameter will only take effect when the setting is saved
ttsc_request.set_saved_path(path)
```

### Step 6 TTS test

Code：

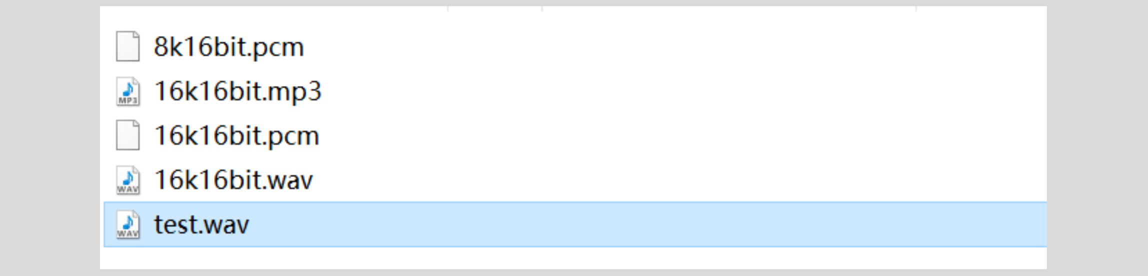
```
# Send a request and return the result. You can view the saved audio in the specified path.
result = ttsc_client.get_ttsc_response(ttsc_request)
print(json.dumps(result, indent=2, ensure_ascii=False))
```

**Result:**

```
{
    "result": {
"data": "UklGRuT···
···
},
    "trace_id": "b9295ebb-1c9c-4d00-b2e9-7d9f3dd63727",
    "is_saved": true,
    "saved_path": "data/test.wav"
}
```

trace_id indicates the internal token of the service, which can be used to trace the specific process in logs. This field is unavailable when the invocation fails. In some error cases, this token string may not be available. result: indicates the recognition result if the invoking is successful. This field is unavailable if the invoking fails. data indicates audio data, which is returned in Base64 encoding format.

The saved speech data is as follows:



8k16bit.pcm
16k16bit.mp3
16k16bit.pcm
16k16bit.wav
test.wav

**Figure 2-3 The Saved Speech Data**

## 2.5 Summary

This chapter mainly introduces the specific operations of using the Speech Interaction Service on Huawei's public cloud to carry out experiments. It mainly implements related functions by issuing RestFul requests through the SDK. When using the SDK to issue RestFul requests, you need to use the necessary tools The configuration of user authentication information is mainly introduced and explained on the system for AK\SK in this chapter, which helps trainees to use speech synthesis to provide practical operation guidance.

# 3 Speech Recognition Based on Seq2Seq

## 3.1 Introduction

### 3.1.1 About this lab

The RNN is suitable for modeling data of the sequence type. Audio data is of this type. Therefore, compared with images, the RNN is better adapted to audio data of this sequence type to recognize audio. Seq2Seq uses the RNN series models and becomes a unique model structure, which is suitable for the scenario where the input is a sequence and the output is also a sequence.

### 3.1.2 Objectives

Upon completion of this task, you will be able to:

- Have a good command of building the Seq2Seq model by using Keras in TensorFlow2.0.
- Have a good command of using the Seq2Seq model to recognize voices.

### 3.1.3 Knowledge Required

This experiment requires knowledge in three aspects:

- The theoretical basis of Seq2Seq is available.
- Keras programming is supported.
- Basic programming in Python.

## 3.2 Procedure

This chapter is based on the Wave framework. The main steps are as follows:

- Read and preprocess data.
- Create a Seq2Seq model, train and test it.

Step 1 Import related modules

Code:

```
#coding=utf-8
import warnings
warnings.filterwarnings("ignore")
import time
import tensorflow as tf
import scipy.io.wavfile as wav
import numpy as np
from six.moves import xrange as range
from python_speech_features import mfcc
from tensorflow.keras.layers import Input,LSTM,Dense
from tensorflow.keras.models import Model,load_model
import pandas as pd
import numpy as np
```

Step 2 Configure data path

Code:

```
audio_filename = "data/audio.wav"
target_filename = "data/label.txt"
```

Step 3 Read data and perform feature extraction

Code:

```
def sparse_tuple_from(sequences, dtype=np.int32):
    indices = []
    values = []

    for n, seq in enumerate(sequences):
        indices.extend(zip([n]*len(seq), range(len(seq))))
        values.extend(seq)

    indices = np.asarray(indices, dtype=np.int64)
    values = np.asarray(values)
    shape = np.asarray([len(sequences), np.asarray(indices).max(0)[1]+1], dtype=np.int64)

    return indices, values, shape

def get_audio_feature():
    # Read the content of the wav file, fs is the sampling rate, audio_filename is the data
    fs, audio = wav.read(audio_filename)

    #Extract mfcc features
    inputs = mfcc(audio, samplerate=fs)
    #Standardize characteristic data, subtract the mean divided by the standard deviation
    feature_inputs = np.asarray(inputs[np.newaxis, :])
    feature_inputs = (feature_inputs - np.mean(feature_inputs))/np.std(feature_inputs)

    # Characteristic data sequence length
    feature_seq_len = [feature_inputs.shape[1]]
    return feature_inputs, feature_seq_len
feature_inputs, feature_seq_len = get_audio_feature()

def get_audio_label():
    with open(target_filename, 'r') as f:
        # The original text is “i like you , do you like me”
        line = f.readlines()[0].strip()
    # Put it in the list, replace the space with ' '
    #['i', ' ', 'like', ' ', 'you',' ', ',',' ', 'do', ' ', 'you', ' ', 'like', ' ', 'me']
    targets = line.split(' ')
    targets.insert(0,'<START>')
    targets.append("<END>")
    print(targets)
    # Convert the list into sparse triples
    train_targets = sparse_tuple_from([targets])
    return targets,train_targets
line_targets,train_traget=get_audio_label()

Result:
['<START>', 'i', 'like', 'you', ',', 'do', 'you', 'like', 'me', '<END>']
```

## Step 4 Configure neural network parameters

Code:

```
target_characters = list(set(line_targets))
INUPT_LENGTH = feature_inputs.shape[-2]
OUTPUT_LENGTH = train_traget[-1][-1]
INPUT_FEATURE_LENGTH = feature_inputs.shape[-1]
OUTPUT_FEATURE_LENGTH = len(target_characters)
N_UNITS = 256
BATCH_SIZE = 1
EPOCH = 20
NUM_SAMPLES = 1
target_texts = []
target_texts.append(line_targets)
```

## Step 5 Create Seq2Seq model

Code：

```
def create_model(n_input,n_output,n_units):
    #encoder
    encoder_input = Input(shape = (None, n_input))
    # The input dimension n_input is the dimension of the input xt at each time step
    encoder = LSTM(n_units, return_state=True)
    # n_units is the number of neurons in each gate in the LSTM unit, and only when return_state is
#set to True will it return to the last state h, c
    _,encoder_h,encoder_c = encoder(encoder_input)
    encoder_state = [encoder_h,encoder_c]
    #Keep the final state of the encoder as the initial state of the decoder
    #decoder
    decoder_input = Input(shape = (None, n_output))
    #The input dimension of decoder is the number of characters
    decoder = LSTM(n_units,return_sequences=True, return_state=True)
    # When training the model, the output sequence of the decoder is required to compare and
#optimize the result, so return_sequences should also be set to True
    decoder_output, _, _ = decoder(decoder_input,initial_state=encoder_state)
    #In the training phase, only the output sequence of the decoder is used, and the final state h.c is
#not required
    decoder_dense = Dense(n_output,activation='softmax')
    decoder_output = decoder_dense(decoder_output)
    # The output sequence passes through the fully connected layer to get the result
    #Generated training model
    model = Model([encoder_input,decoder_input],decoder_output)
    # The first parameter is the input of the training model, including the input of encoder and
#decoder, and the second parameter is the output of the model, including the output of the decoder
    # Inference stage, used in the prediction process
    # Inference model—encoder
    encoder_infer = Model(encoder_input,encoder_state)

    # Inference model -decoder
    decoder_state_input_h = Input(shape=(n_units,))
    decoder_state_input_c = Input(shape=(n_units,))
    # The state of the last moment h,c
    decoder_state_input = [decoder_state_input_h, decoder_state_input_c]

    decoder_infer_output, decoder_infer_state_h, decoder_infer_state_c =
decoder(decoder_input,initial_state=decoder_state_input)
    #The current state
    decoder_infer_state = [decoder_infer_state_h, decoder_infer_state_c]
    decoder_infer_output = decoder_dense(decoder_infer_output)# Current time output
    decoder_infer =
Model([decoder_input]+decoder_state_input,[decoder_infer_output]+decoder_infer_state)

    return model, encoder_infer, decoder_infer
model_train, encoder_infer, decoder_infer = create_model(INPUT_FEATURE_LENGTH,
OUTPUT_FEATURE_LENGTH, N_UNITS)
model_train.compile(optimizer='adam', loss='categorical_crossentropy')
model_train.summary()
```

**Result:**

```
Model: "model"
__________________________________________________________________________
Layer (type)                  Output Shape          Param #     Connected to
==================================================================
input_1 (InputLayer)          (None, None, 13)      0
__________________________________________________________________________
input_2 (InputLayer)          (None, None, 8)       0
__________________________________________________________________________
lstm_1 (LSTM)                 [(None, 256), (None, 276480       input_1[0][0]
__________________________________________________________________________
lstm_2 (LSTM)                 [(None, None, 256),   271360      input_2[0][0]
                                                                lstm_1[0][1]
                                                                lstm_1[0][2]
__________________________________________________________________________
dense_1 (Dense)               (None, None, 8)       2056        lstm_2[0][0]
==================================================================
Total params: 549,896
Trainable params: 549,896
```

```
Non-trainable params: 0
_________________________________________________________________
```

## Step 6 Configure training data

Code:

```
encoder_input = feature_inputs
decoder_input = np.zeros((NUM_SAMPLES,OUTPUT_LENGTH,OUTPUT_FEATURE_LENGTH))
decoder_output = np.zeros((NUM_SAMPLES,OUTPUT_LENGTH,OUTPUT_FEATURE_LENGTH))
target_dict = {char:index for index,char in enumerate(target_characters)}
target_dict_reverse = {index:char for index,char in enumerate(target_characters)}

print(decoder_input.shape)
for seq_index,seq in enumerate(target_texts):

    for char_index,char in enumerate(seq):
        print(char_index,char)
        decoder_input[seq_index,char_index,target_dict[char]] = 1.0
        if char_index > 0:
            decoder_output[seq_index,char_index-1,target_dict[char]] = 1.0
```

**Result:**

```
(1, 10, 8)
0 <START>
1 i
2 like
3 you
4 ,
5 do
6 you
7 like
8 me
9 <END>
```

## Step 7 Model training

Code:

```
#Get training data, in this example only one sample of training data
model_train.fit([encoder_input,decoder_input],decoder_output,batch_size=BATCH_SIZE,epochs=EPOC
H,validation_split=0)
```

**Result:**

```
Train on 1 samples
Epoch 1/20
1/1 [==============================] - 6s 6s/sample - loss: 1.6983
Epoch 2/20
1/1 [==============================] - 0s 464ms/sample - loss: 1.6155
Epoch 3/20
1/1 [==============================] - 1s 502ms/sample - loss: 1.5292
```

```
Epoch 4/20
1/1 [==============================] - 0s 469ms/sample - loss: 1.4335
Epoch 5/20
1/1 [==============================] - 1s 520ms/sample - loss: 1.3506
Epoch 6/20
1/1 [==============================] - 0s 445ms/sample - loss: 1.2556
Epoch 7/20
1/1 [==============================] - 0s 444ms/sample - loss: 1.1671
Epoch 8/20
1/1 [==============================] - 0s 424ms/sample - loss: 1.0965
Epoch 9/20
1/1 [==============================] - 0s 432ms/sample - loss: 1.0321
Epoch 10/20
1/1 [==============================] - 0s 448ms/sample - loss: 0.9653
Epoch 11/20
1/1 [==============================] - 1s 501ms/sample - loss: 0.9038
Epoch 12/20
1/1 [==============================] - 0s 471ms/sample - loss: 0.8462
Epoch 13/20
1/1 [==============================] - 0s 453ms/sample - loss: 0.7752
Epoch 14/20
1/1 [==============================] - 0s 444ms/sample - loss: 0.7188
Epoch 15/20
1/1 [==============================] - 0s 452ms/sample - loss: 0.6608
Epoch 16/20
1/1 [==============================] - 0s 457ms/sample - loss: 0.6058
Epoch 17/20
1/1 [==============================] - 1s 522ms/sample - loss: 0.5542
Epoch 18/20
1/1 [==============================] - 0s 444ms/sample - loss: 0.5001
Epoch 19/20
1/1 [==============================] - 0s 433ms/sample - loss: 0.4461
Epoch 20/20
1/1 [==============================] - 0s 432ms/sample - loss: 0.4020
<tensorflow.python.keras.callbacks.History at 0x1ecacdec128>
```

### Step 8 Model testing

Code：

```
def predict_chinese(source,encoder_inference, decoder_inference, n_steps, features):
    # First obtain the hidden state of the predicted input sequence through the inference encoder
    state = encoder_inference.predict(source)
    # The first character'\t' is the starting mark
    predict_seq = np.zeros((1,1,features))
    predict_seq[0,0,target_dict['<START>']] = 1

    output = ''
    # Start to predict about the hidden state obtained by the encoder
    # Each cycle uses the last predicted character as input to predict the next character until the
#terminator is predicted
    for i in range(n_steps):#n_steps is maximum sentence length
        # Input the hidden state of h, c at the last moment to the decoder, and the predicted
#character predict_seq of the last time
        yhat,h,c = decoder_inference.predict([predict_seq]+state)
        # Note that yhat here is the result output after Dense, so it is different from h
        char_index = np.argmax(yhat[0,-1,:])
        char = target_dict_reverse[char_index]
#          print(char)

        state = [h,c] # This state will continue to be passed as the next initial state
        predict_seq = np.zeros((1,1,features))
        predict_seq[0,0,char_index] = 1
        if char == '<END>': # Stop when the terminator is predicted
            break
        output +=" " +char
return output
out = 
predict_chinese(encoder_input,encoder_infer,decoder_infer,OUTPUT_LENGTH,OUTPUT_FEATURE_LEN
GTH)
print(out)
```

**Result:**

```
i like you , do you like me
```

This experiment only uses one training sample. Interested students can further expand the model to train on more sample spaces. In addition, the model obtained during each training may have different output results during prediction due to different

## 3.3 Summary

The main content of this experiment is based on Python and scipy, python_speech_features, six, keras, and TensorFlow frameworks to recognize speech data through Seq2Seq. After the experiment, trainees can master the construction of Seq2Seq model through Keras and the application of Seq2Seq model to speech recognition.

Huawei AI Certification Training

# HCIP-AI-EI Developer

# Natural Language Processing Lab Guide

ISSUE:2.0

HUAWEI TECHNOLOGIES CO., LTD.

**Trademarks and Permissions**

**Notice**

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

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

**Huawei Technologies Co., Ltd.**

Address: Huawei Industrial Base Bantian, Longgang Shenzhen 518129
People's Republic of China

Website: http://e.huawei.com

# Huawei Certificate System

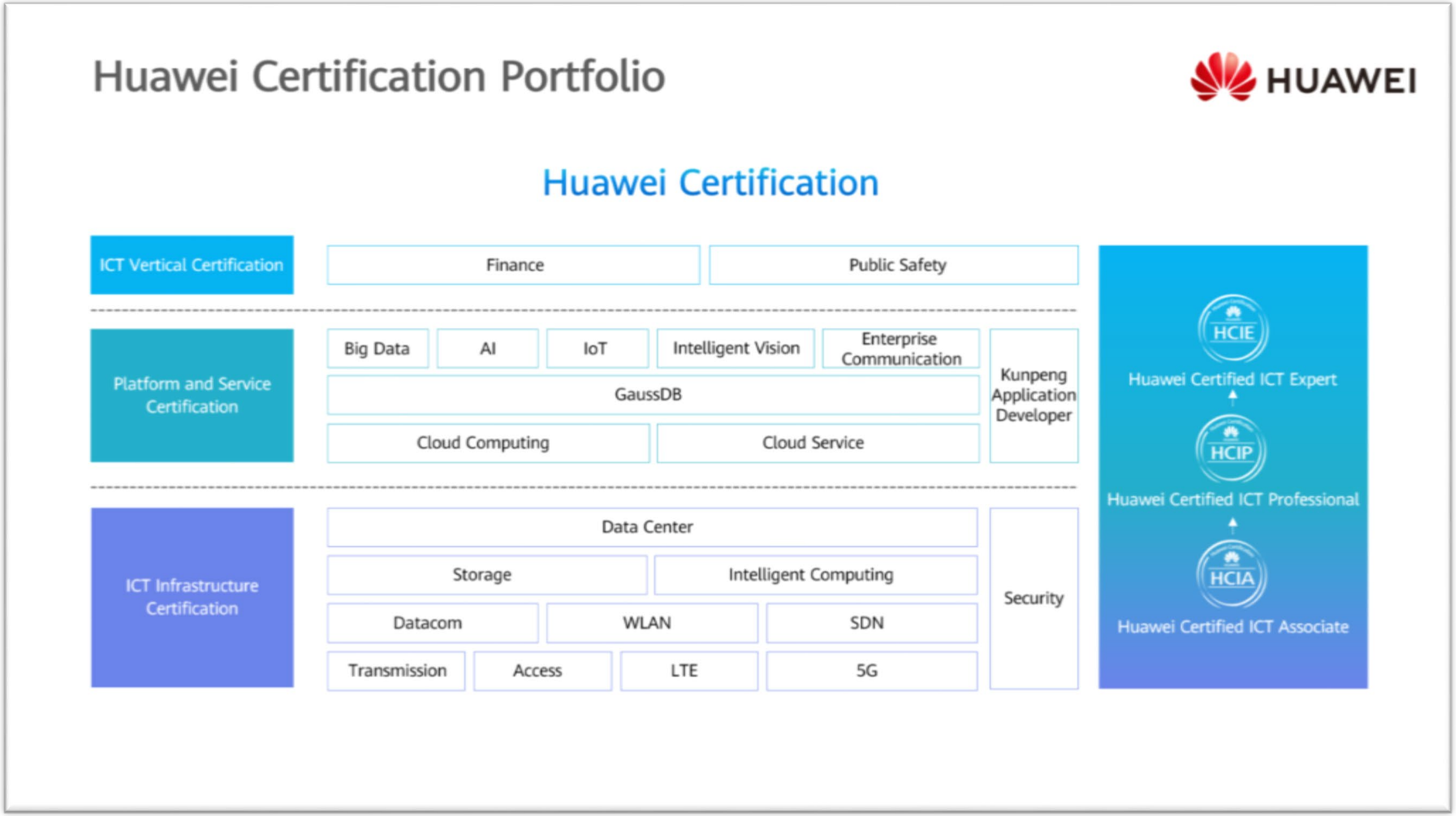
Huawei's certification system is the industry's only one that covers all ICT technical fields. It is developed relying on Huawei's 'platform + ecosystem' strategy and new ICT technical architecture featuring cloud-pipe-device synergy. It provides three types of certifications: ICT Infrastructure Certification, Platform and Service Certification, and ICT Vertical Certification.

To meet ICT professionals' progressive requirements, Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

HCIP-AI-EI Developer V2.0 certification is intended to cultivate professionals who have acquired basic theoretical knowledge about image processing, speech processing, and natural language processing and who are able to conduct development and innovation using Huawei enterprise AI solutions (such as HUAWEI CLOUD EI), general open-source frameworks, and ModelArts, a one-stop development platform for AI developers.

The content of HCIP-AI-EI Developer V2.0 certification includes but is not limited to: neural network basics, image processing theory and applications, speech processing theory and applications, natural language processing theory and applications, ModelArts overview, and image processing, speech processing, natural language processing, and ModelArts platform development experiments. ModelArts is a one-stop development platform for AI developers. With data preprocessing, semi-automatic data labeling, large-scale distributed training, automatic modeling, and on-demand model deployment on devices, edges, and clouds, ModelArts helps AI developers build models quickly and manage the lifecycle of AI development. Compared with V1.0, HCIP-AI-EI Developer V2.0 adds the ModelArts overview and development experiments. In addition, some new EI cloud services are updated.

HCIP-AI-EI Developer V2.0 certification proves that you have systematically understood and mastered neural network basics, image processing theory and applications, speech processing theory and applications, ModelArts overview, natural language processing theory and applications, image processing application development, speech processing application development, natural language processing application development, and ModelArts platform development. With this certification, you will acquire (1) the knowledge and skills for AI pre-sales technical support, AI after-sales technical support, AI product sales, and AI project management; (2) the ability to serve as an image processing developer, speech processing developer, or natural language processing developer.



Huawei Certification Portfolio

# About This Document

## Overview

This document is a training course for HCIP-AI certification. It is intended for trainees who are going to take the HCIP-AI exam or readers who want to understand basic AI knowledge. After mastering this lab, you can use the Python SDK to call NLP APIs of HUAWEI CLOUD EI or use ModelArts to build and train your NLP algorithm models.

## Description

This lab consists of three groups of experiments, involving basic algorithms for natural language processing, natural language understanding, and natural language generation.

- Experiment 1: HUAWEI CLOUD EI Natural Language Processing Service
- Experiment 2: Text classification
- Experiment 3: Machine Translation

## Background Knowledge Required

This course is a basic course for Huawei certification. To better master the contents of this course, readers must:

- Basic Python language editing capability
- Have a certain theoretical basis for natural language processing.
- Understand the TensorFlow framework.

## Experiment Environment Overview

- ModelArts TensorFlow-2.1.0 8-core 32 g CPU environment

# Contents

# 1 HUAWEI CLOUD EI Natural Language Processing Service

## 1.1 Introduction

Natural Language Processing (NLP) is artificial intelligence technologies for text analysis and mining. HUAWEI CLOUD provide the NLP services aim to help users efficiently process text.

NLP consists of the following subservices, but most services are only support Chineses language:

Natural Language Processing Fundamentals (NLPF) provides APIs related to natural languages, such as word segmentation, naming entity recognition (NER), keyword extraction, and short text similarity, it can be used in scenarios such as intelligent Q&A, chatbot, public opinion analysis, content recommendation, and e-commerce evaluation analysis.

Language Generation (LG) provides APIs related to language generation for users, such as text abstracts. It can be used in scenarios such as news abstract generation, document abstract generation, search result fragment generation, and commodity review abstract.

Language Understanding (LU) provides APIs related to language understanding, such as text classification and emotion analysis, and can be used in scenarios such as emotion analysis, content detection, and advertisement recognition.

## 1.2 Objective

This experiment describes how to use NLP services in HUAWEI CLOUD. Currently, HUAWEI CLOUD provides the Python SDK for NLP. This experiment will guide trainees to understand and master how to use the Python SDK to call NLP services.

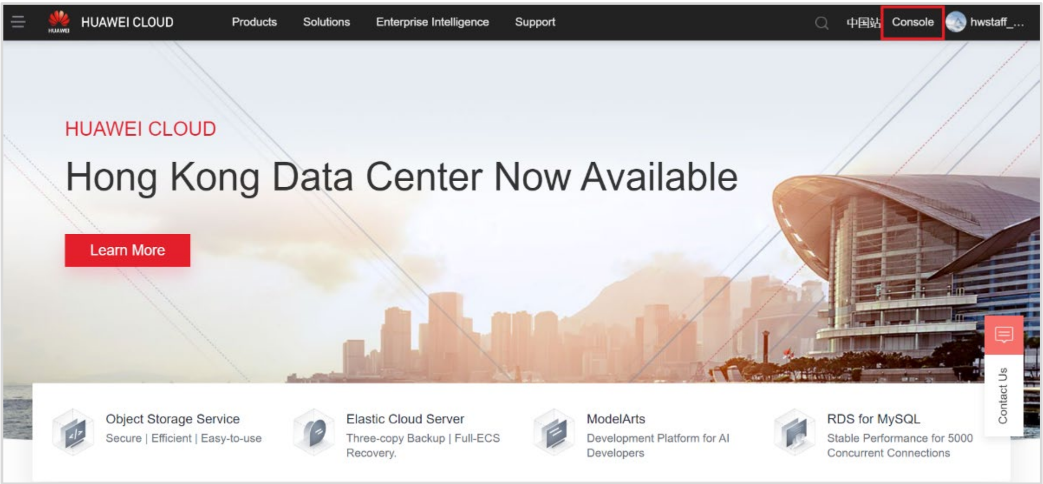
## 1.3 Procedure

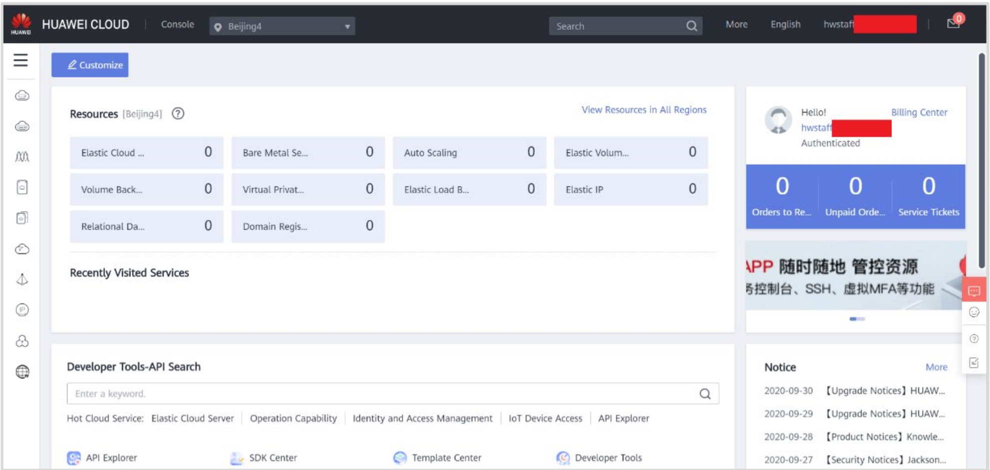
In this experiment, you need to download the NLP SDK from HUAWEI CLOUD and access the service in two ways: AK/SK information is used for identity authentication, and the underlying API service of the SDK is invoked to submit a RESTful service request. Token information of a user is used to submit a RESTful request. The procedure is as follows:

## 1.3.1 Preparing the Experiment Environment

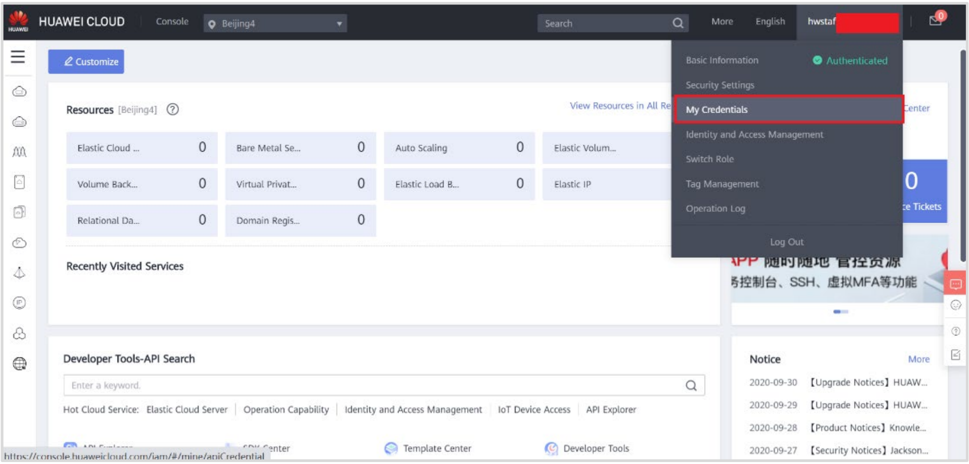
### 1.3.1.1 Obtain the project code

Step 1 Register and log in to the console.

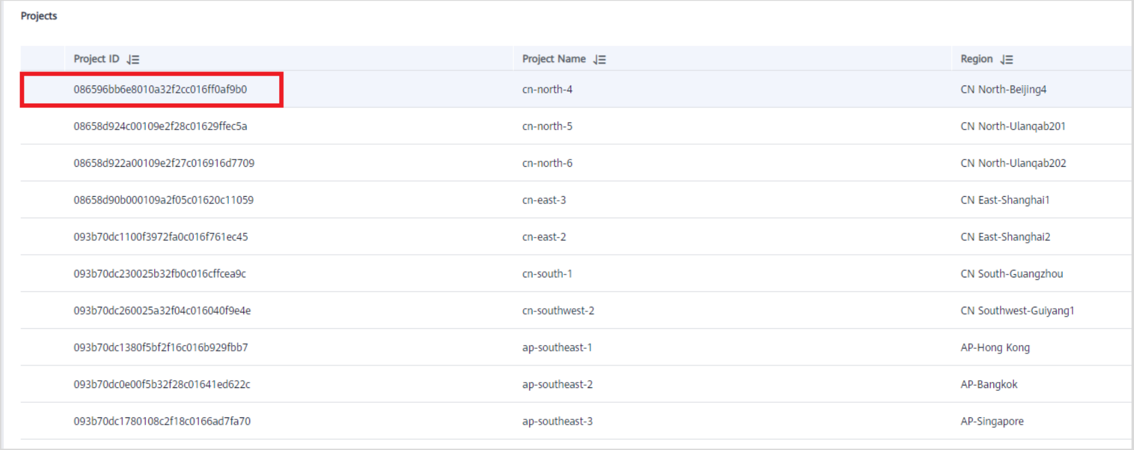

HUAWEI CLOUD
Products
Solutions
Enterprise Intelligence
Support
中国站
Console
hwstaff_...
HUAWEI CLOUD
Hong Kong Data Center Now Available
Learn More
Contact Us
Object Storage Service
Secure | Efficient | Easy-to-use
Elastic Cloud Server
Three-copy Backup | Full-ECS Recovery.
ModelArts
Development Platform for AI Developers
RDS for MySQL
Stable Performance for 5000 Concurrent Connections



Step 2 Click the username and select “My Credentials” from the drop-down list.



Step 3 On the My Credential page, view the project ID in the projects list.

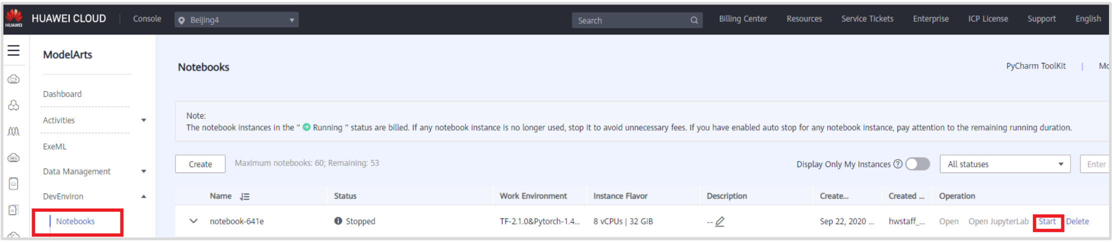


Projects

| Project ID | Project Name | Region |
|---|---|---|
| 086596bb6e8010a32f2cc016ff0af9b0 | cn-north-4 | CN North-Beijing4 |
| 08658d924c00109e2f28c01629ffec5a | cn-north-5 | CN North-Ulanqab201 |
| 08658d922a00109e2f27c016916d7709 | cn-north-6 | CN North-Ulanqab202 |
| 08658d90b000109a2f05c01620c11059 | cn-east-3 | CN East-Shanghai1 |
| 093b70dc1100f3972fa0c016f761ec45 | cn-east-2 | CN East-Shanghai2 |
| 093b70dc230025b32fb0c016cffcea9c | cn-south-1 | CN South-Guangzhou |
| 093b70dc260025a32f04c016040f9e4e | cn-southwest-2 | CN Southwest-Guiyang1 |
| 093b70dc1380f5bf2f16c016b929fbb7 | ap-southeast-1 | AP-Hong Kong |
| 093b70dc0e00f5b32f28c01641ed622c | ap-southeast-2 | AP-Bangkok |
| 093b70dc1780108c2f18c0166ad7fa70 | ap-southeast-3 | AP-Singapore |

## 1.3.1.2 Download and Use the SDK

Step 1 Go to the created notebook environment with the 8-core 32 GB modelArts TensorFlow2.1.0 configuration.



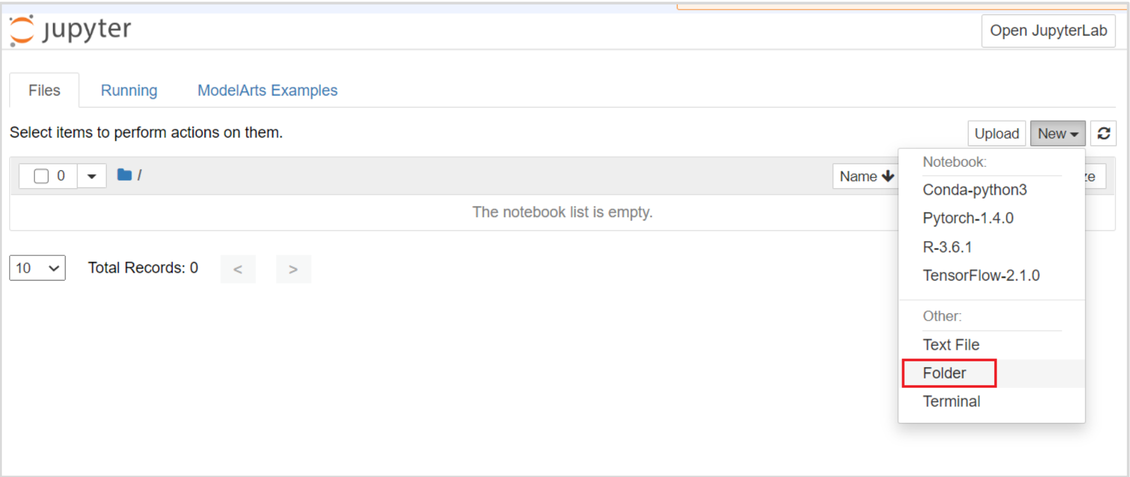
Notebooks

Note:
The notebook instances in the " Running " status are billed. If any notebook instance is no longer used, stop it to avoid unnecessary fees. If you have enabled auto stop for any notebook instance, pay attention to the remaining running duration.

Maximum notebooks: 60; Remaining: 53

| Name | Status | Work Environment | Instance Flavor | Description | Create... | Created ... | Operation |
|---|---|---|---|---|---|---|---|
| notebook-641e | Stopped | TF-2.1.0&Pytorch-1.4... | 8 vCPUs \| 32 GiB | -- | Sep 22, 2020 ... | hwstaff_... | Open Open JupyterLab Start Delete |

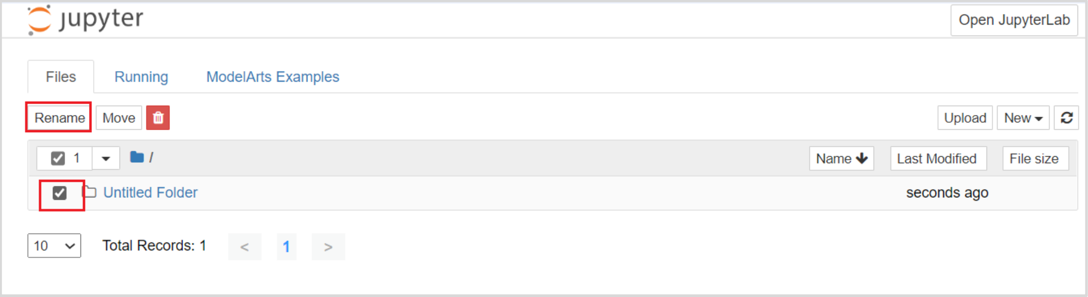
Step 2 Go to the notebook page, create a folder, and rename the folder “huawei_cloud_ei”.

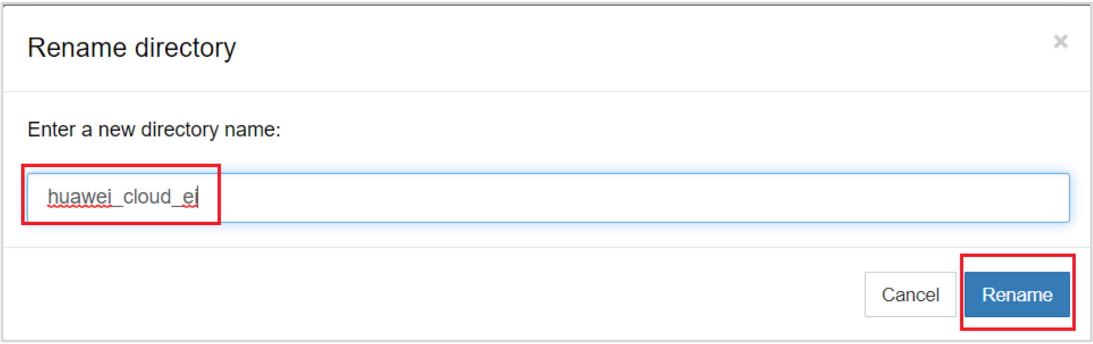


Select items to perform actions on them.

The notebook list is empty.

Total Records: 0

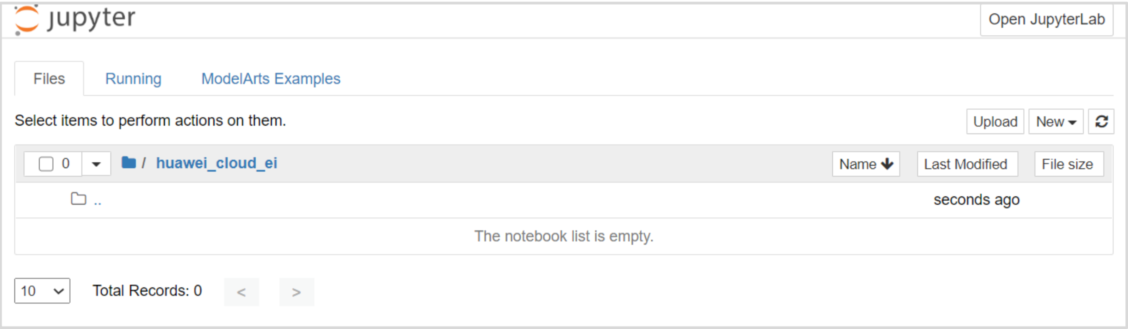



Rename directory

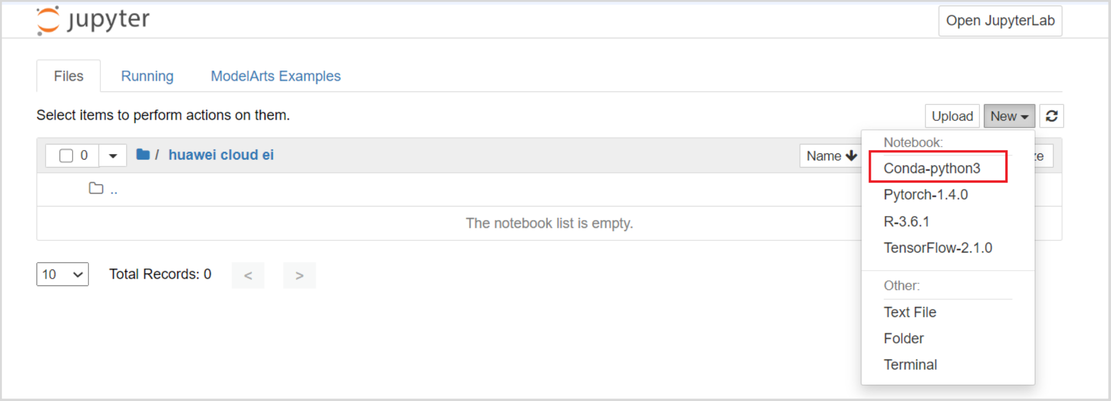
Enter a new directory name:

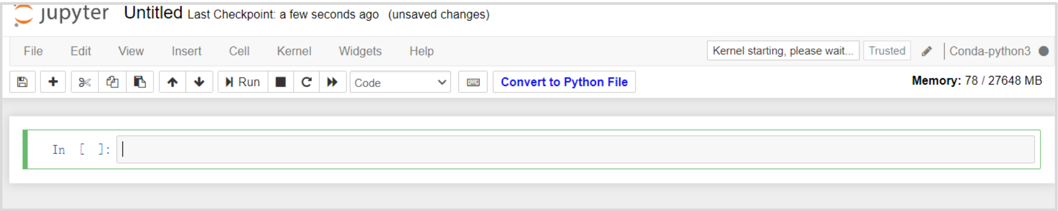
huawei_cloud_ei

Step 3 Click the newly created huawei_cloud_ei folder.



Step 4 Create a notebook file and select the conda-python3 environment.



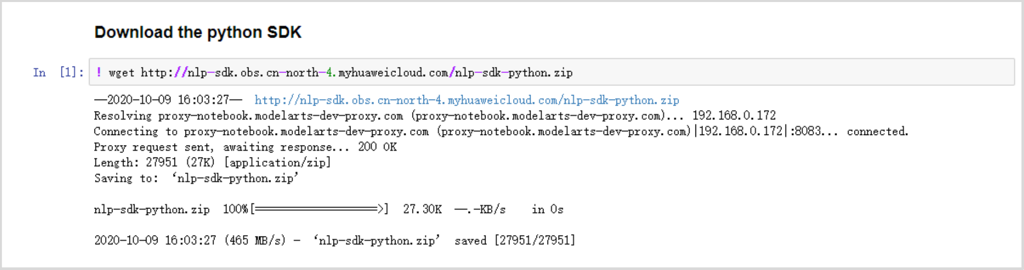

jupyter Untitled Last Checkpoint: a few seconds ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Kernel starting, please wait...
Trusted
Conda-python3
Run
Code
Convert to Python File
Memory: 78 / 27648 MB
In [ ]:

### Step 5 Download the Python SDK

Input:

```
! wget https://huawei-ai-certification.obs.cn-north-4.myhuaweicloud.com/ENG/HCIP-AI%20EI%20Developer/V2.1/nlp-sdk-python.zip
```

Output:



Download the python SDK

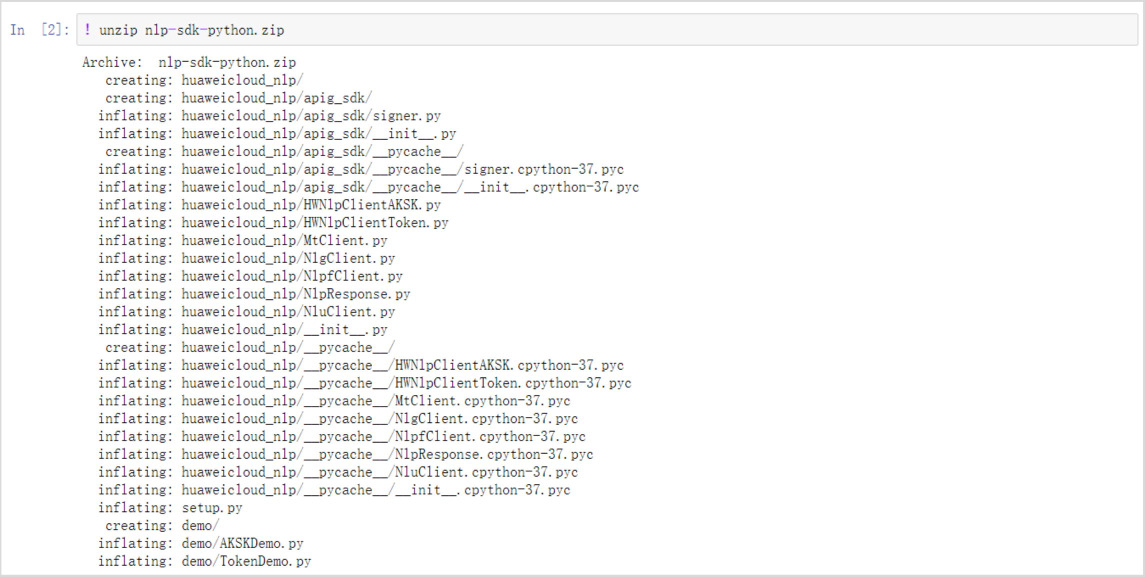
```
In  [1]: ! wget http://nlp-sdk.obs.cn-north-4.myhuaweicloud.com/nlp-sdk-python.zip

         —2020-10-09 16:03:27—  http://nlp-sdk.obs.cn-north-4.myhuaweicloud.com/nlp-sdk-python.zip
         Resolving proxy-notebook.modelarts-dev-proxy.com (proxy-notebook.modelarts-dev-proxy.com)... 192.168.0.172
         Connecting to proxy-notebook.modelarts-dev-proxy.com (proxy-notebook.modelarts-dev-proxy.com)|192.168.0.172|:8083... connected.
         Proxy request sent, awaiting response... 200 OK
         Length: 27951 (27K) [application/zip]
         Saving to: ‘nlp-sdk-python.zip’

         nlp-sdk-python.zip  100%[==================>]  27.30K  —.-KB/s    in 0s

         2020-10-09 16:03:27 (465 MB/s) - ‘nlp-sdk-python.zip’  saved [27951/27951]
```

### Step 6 Decompressing the SDK

Input:

```
! unzip nlp-sdk-python.zip
```

Output:



```
In  [2]: ! unzip nlp-sdk-python.zip

         Archive:  nlp-sdk-python.zip
            creating: huaweicloud_nlp/
            creating: huaweicloud_nlp/apig_sdk/
           inflating: huaweicloud_nlp/apig_sdk/signer.py
           inflating: huaweicloud_nlp/apig_sdk/__init__.py
            creating: huaweicloud_nlp/apig_sdk/__pycache__/
           inflating: huaweicloud_nlp/apig_sdk/__pycache__/signer.cpython-37.pyc
           inflating: huaweicloud_nlp/apig_sdk/__pycache__/__init__.cpython-37.pyc
           inflating: huaweicloud_nlp/HWNlpClientAKSK.py
           inflating: huaweicloud_nlp/HWNlpClientToken.py
           inflating: huaweicloud_nlp/MtClient.py
           inflating: huaweicloud_nlp/NlgClient.py
           inflating: huaweicloud_nlp/NlpfClient.py
           inflating: huaweicloud_nlp/NlpResponse.py
           inflating: huaweicloud_nlp/NluClient.py
           inflating: huaweicloud_nlp/__init__.py
            creating: huaweicloud_nlp/__pycache__/
           inflating: huaweicloud_nlp/__pycache__/HWNlpClientAKSK.cpython-37.pyc
           inflating: huaweicloud_nlp/__pycache__/HWNlpClientToken.cpython-37.pyc
           inflating: huaweicloud_nlp/__pycache__/MtClient.cpython-37.pyc
           inflating: huaweicloud_nlp/__pycache__/NlgClient.cpython-37.pyc
           inflating: huaweicloud_nlp/__pycache__/NlpfClient.cpython-37.pyc
           inflating: huaweicloud_nlp/__pycache__/NlpResponse.cpython-37.pyc
           inflating: huaweicloud_nlp/__pycache__/NluClient.cpython-37.pyc
           inflating: huaweicloud_nlp/__pycache__/__init__.cpython-37.pyc
           inflating: setup.py
            creating: demo/
           inflating: demo/AKSKDemo.py
           inflating: demo/TokenDemo.py
```

## 1.3.2 NLP Basic Service

### Step 1 Importing SDKs

Input:

```
import json
from huaweicloud_nlp.MtClient import MtClient
from huaweicloud_nlp.NlpfClient import NlpfClient
from huaweicloud_nlp.NluClient import NluClient
from huaweicloud_nlp.NlgClient import NlgClient
from huaweicloud_nlp.HWNlpClientToken import HWNlpClientToken
import warnings
warnings.filterwarnings("ignore")
```

Step 2 Token authentication

Input:

```
tokenClient = HWNlpClientToken("domain_name", "user_name", "your_password", "cn-north-4", "your_project_id")
```

The token authentication mode is used. You need to enter the domain account name, user name, password, region, and project ID.

Step 3 Initializing the Client

Input:

```
nlpfClient = NlpfClient(tokenClient)
```

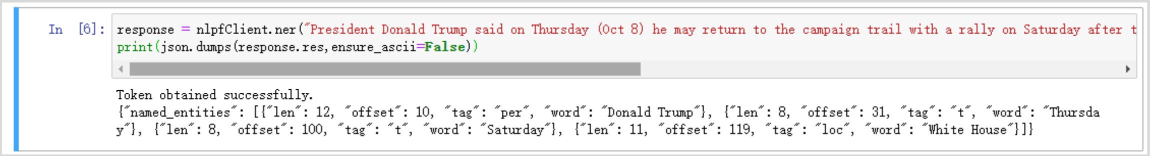
Step 4 Named Entity Recognition (Basic Edition)

This API is used for named entity recognition (NER). Currently, it can be called to identify and analyze person names, locations, time, and organization names in the text.

Input:

```
response = nlpfClient.ner("President Donald Trump said on Thursday (Oct 8) he may return to the campaign trail with a rally on Saturday after the White House physician said he had completed his course of therapy for the novel coronavirus and could resume public events.", "en")
print(json.dumps(response.res,ensure_ascii=False))
```

Output:



```
In  [6]: response = nlpfClient.ner("President Donald Trump said on Thursday (Oct 8) he may return to the campaign trail with a rally on Saturday after t
         print(json.dumps(response.res, ensure_ascii=False))

         Token obtained successfully.
         {"named_entities": [{"len": 12, "offset": 10, "tag": "per", "word": "Donald Trump"}, {"len": 8, "offset": 31, "tag": "t", "word": "Thursda
         y"}, {"len": 8, "offset": 100, "tag": "t", "word": "Saturday"}, {"len": 11, "offset": 119, "tag": "loc", "word": "White House"}]}
```

## 1.3.3 Natural Language Generation Service

Step 1 Initializing the Client

Input:

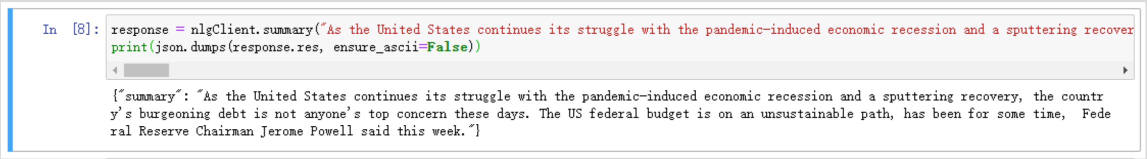
```
nlgClient = NlgClient(tokenClient)
```

### Step 2 Text Summary (Basic)

Input:

```
    response = nlgClient.summary("As the United States continues its struggle with the pandemic-
induced economic recession and a sputtering recovery, the country's burgeoning debt is not anyone's
top concern these days.   Even deficit hawks are urging a dysfunctional Washington and a chaotic
White House to approve another round of badly needed stimulus to the tune of trillions.   The US
federal budget is on an unsustainable path, has been for some time,   Federal Reserve Chairman
Jerome Powell said this week. But, Powell added,   This is not the time to give priority to those
concerns.   However, when the country eventually pulls out of its current health and economic crises,
Americans will be left with a debt hangover. On Thursday, the Congressional Budget Office estimated
that for fiscal year 2020, which ended September 30, the US deficit hit $3.13 trillion -- or 15.2% of
GDP -- thanks to the chasm between what the country spent ($6.55 trillion) and what it took in
($3.42 trillion) for the year. As a share of the economy, the estimated 2020 deficit is more than triple
what the annual deficit was in 2019. And it's the highest it has been since just after World War II. The
reason for the huge year-over-year jump is simple: Starting this spring, the federal government spent
more than $4 trillion to help stem the economic pain to workers and businesses caused by sudden
and widespread business shutdowns. And most people agree more money will need to be spent until
the White House manages to get the Covid-19 crisis under control. The Treasury Department won't
put out final numbers for fiscal year 2020 until later this month. But if the CBO's estimates are on the
mark, the country's total debt owed to investors -- which is essentially the sum of annual deficits that
have accrued over the years -- will have outpaced the size of the economy, coming in at nearly 102%
of GDP, according to calculations from the Committee for a Responsible Federal Budget. The debt
hasn't been that high since 1946, when the federal debt was 106.1% of GDP.   Debt is the size of the
economy today, and soon it will be larger than any time in history,   CRFB president Maya
MacGuineas said. The problem with such high debt levels going forward is that they will increasingly
constrain what the government can do to meet the country's needs. Spending is projected to continue
rising and is far outpacing revenue. And interest payments alone on the debt -- even if rates remain
low -- will consume an ever-growing share of tax dollars. Given the risks of future disruptions, like a
pandemic, a debt load that already is outpacing economic growth puts the country at greater risk of
a fiscal crisis, which in turn would require sharp cuts to the services and benefits on which Americans
rely.   There is no set tipping point at which a fiscal crisis becomes likely or imminent, nor is there an
identifiable point at which interest costs as a percentage of GDP become unsustainable,   CBO
director Phillip Swagel said last month.   But as the debt grows, the risks become greater.   ","The US
debt is now projected to be larger than the US economy",None,"en")
    print(json.dumps(response.res, ensure_ascii=False))
```

Output:



```
In  [8]: response = nlgClient.summary("As the United States continues its struggle with the pandemic-induced economic recession and a sputtering recover
         print(json.dumps(response.res, ensure_ascii=False))

         {"summary": "As the United States continues its struggle with the pandemic-induced economic recession and a sputtering recovery, the countr
         y's burgeoning debt is not anyone's top concern these days. The US federal budget is on an unsustainable path, has been for some time,  Fede
         ral Reserve Chairman Jerome Powell said this week."}
```

## 1.4 Experiment Summary

This chapter describes how to use NLP services in HUAWEI CLOUD to perform experiments, including “Named Entity Recognition(NER)” and “Text Summary”.

# 2 Text Classification

## 2.1 Introduction

This chapter describes how to implement a text classification model. The specific task is Sentiment Analysis by user comments. The models include:

- Naive Bayes
- Support Vector Machine
- TextCNN

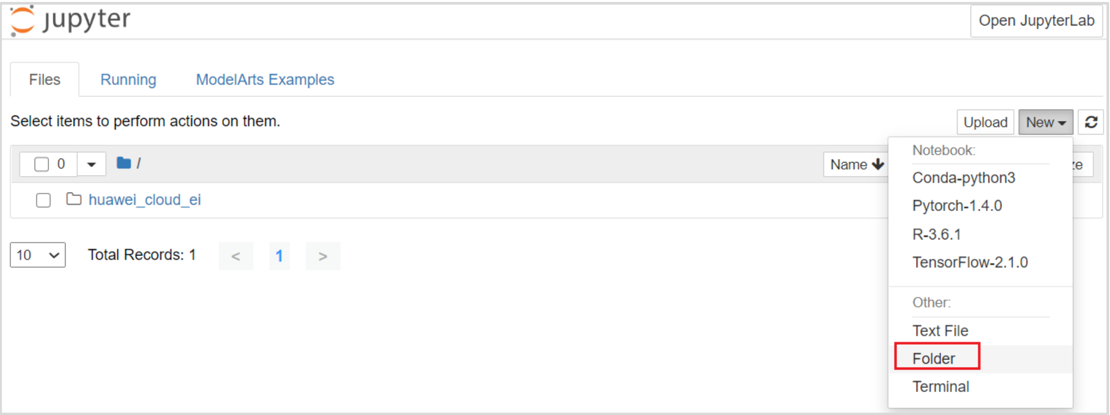
## 2.2 Objective

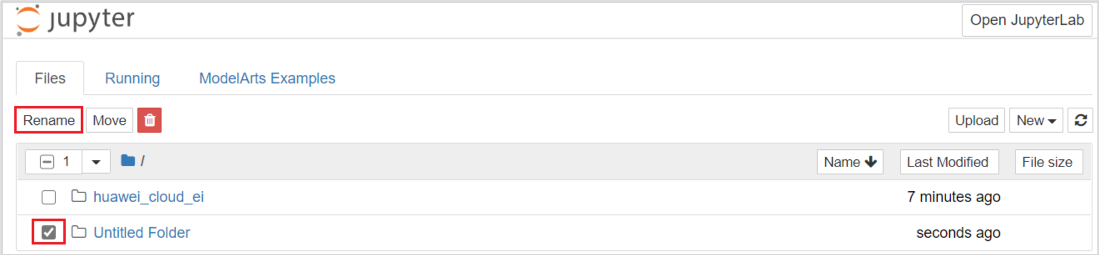
- Understand the basic principles and process of text categorization tasks.
- Understand the differences between Naive Bayes, SVM, and TextCNN algorithms.
- Master the method of building a neural network based on TensorFlow 2.x.

## 2.3 Procedure

### 2.3.1 Environment Preparation

Step 1 Go to the notebook page, create a folder, and rename the folder text_classification.

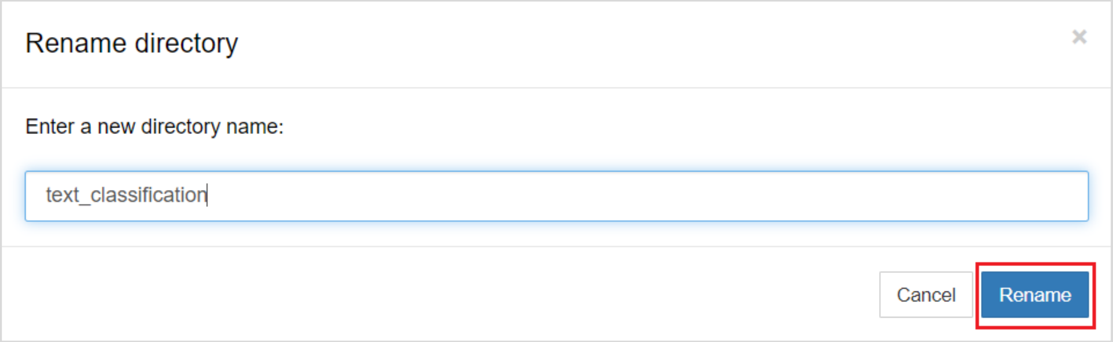




☐ 1 /

☐ huawei_cloud_ei 7 minutes ago

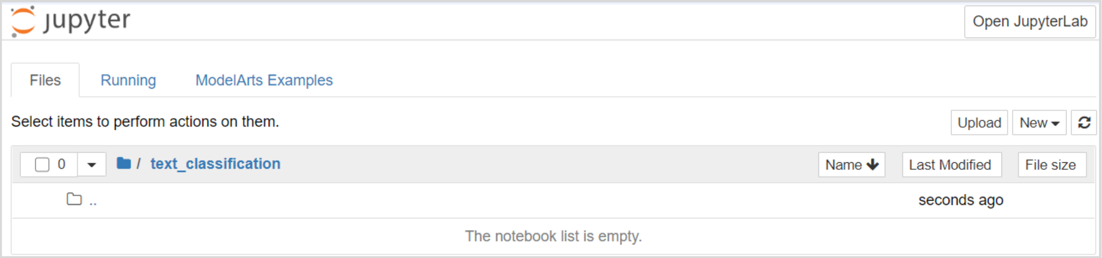
☑ Untitled Folder seconds ago



Enter a new directory name:

text_classification

Step 2 Click the created text_classification folder.



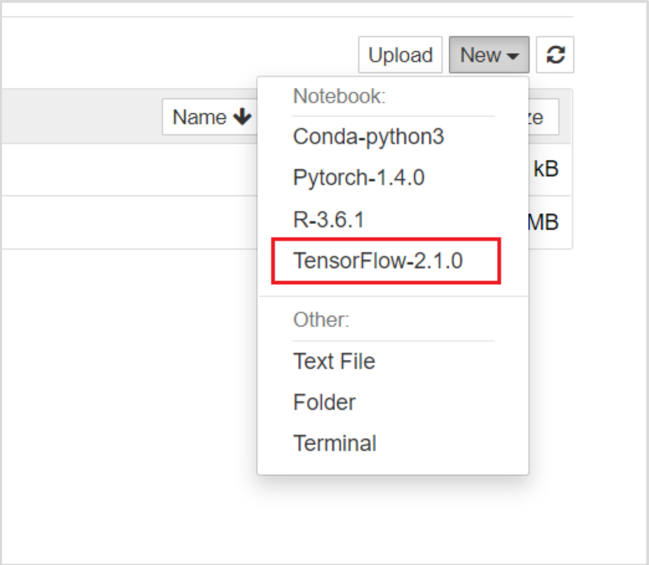
Select items to perform actions on them.

☐ 0 / text_classification

.. seconds ago

The notebook list is empty.

Step 3 Create a notebook file and select the TensorFlow-2.1.0 environment.

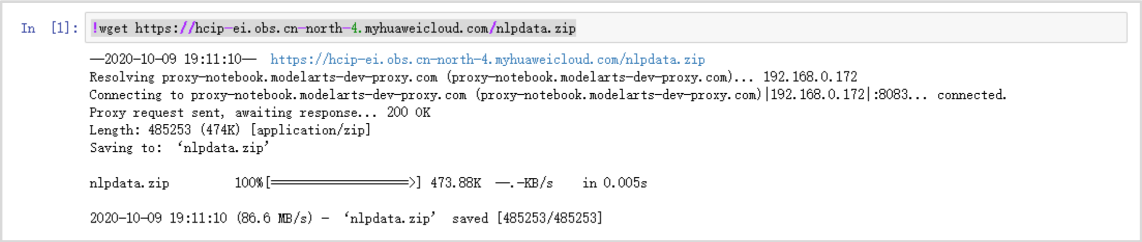


Step 4 Downloading Data

Input:

```
!wget https://huawei-ai-certification.obs.cn-north-4.myhuaweicloud.com/ENG/HCIP-AI%20EI%20Developer/V2.1/nlpdata.zip
```

Output:



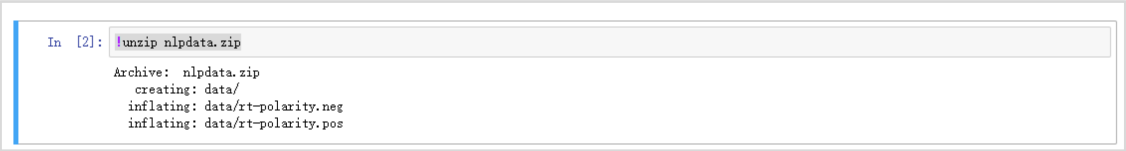
```
In  [1]: !wget https://hcip-ei.obs.cn-north-4.myhuaweicloud.com/nlpdata.zip

         —2020-10-09 19:11:10—  https://hcip-ei.obs.cn-north-4.myhuaweicloud.com/nlpdata.zip
         Resolving proxy-notebook.modelarts-dev-proxy.com (proxy-notebook.modelarts-dev-proxy.com)... 192.168.0.172
         Connecting to proxy-notebook.modelarts-dev-proxy.com (proxy-notebook.modelarts-dev-proxy.com)|192.168.0.172|:8083... connected.
         Proxy request sent, awaiting response... 200 OK
         Length: 485253 (474K) [application/zip]
         Saving to: 'nlpdata.zip'

         nlpdata.zip         100%[=================>] 473.88K  —.-KB/s    in 0.005s

         2020-10-09 19:11:10 (86.6 MB/s) - 'nlpdata.zip'  saved [485253/485253]
```

Step 5 Decompressing Data

Input:

```
!unzip nlpdata.zip
```

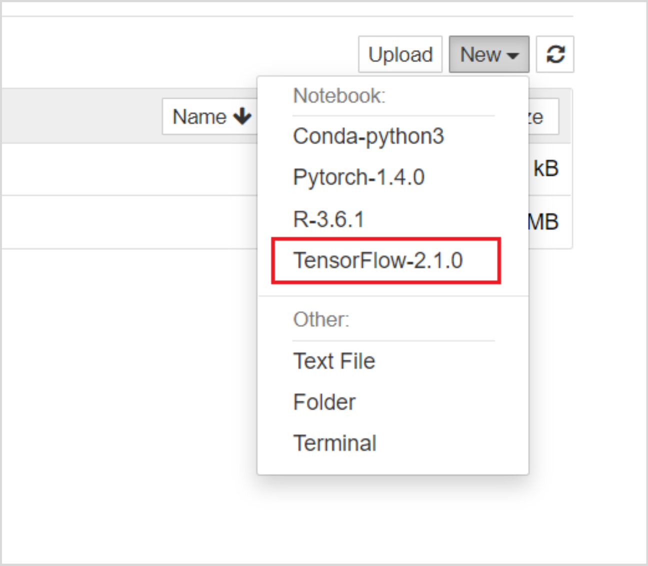
Output:



```
In  [2]: !unzip nlpdata.zip

         Archive:  nlpdata.zip
            creating: data/
           inflating: data/rt-polarity.neg
           inflating: data/rt-polarity.pos
```

## 2.3.2 Naive Bayesian text classification

Step 1 Create a notebook file and select the TensorFlow-2.1.0 environment.



Step 2 Importing Related Library

Input:

```
import re
import pandas as pd
import numpy as np
from sklearn.metrics import classification_report
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report, accuracy_score
```

## Step 3 Data preprocessing

Input:

```
def clean_str(string):
    """
    Tokenization/string cleaning for all datasets except for SST.
    Original taken from https://github.com/yoonkim/CNN_sentence/blob/master/process_data.py
    """
    string = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"\'s", " \'s", string)
    string = re.sub(r"\'ve", " \'ve", string)
    string = re.sub(r"n\'t", " n\'t", string)
    string = re.sub(r"\'re", " \'re", string)
    string = re.sub(r"\'d", " \'d", string)
    string = re.sub(r"\'ll", " \'ll", string)
    string = re.sub(r",", " , ", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", " \( ", string)
    string = re.sub(r"\)", " \) ", string)
    string = re.sub(r"\?", " \? ", string)
    string = re.sub(r"\s{2,}", " ", string)
    return string.strip().lower()


def load_data_and_labels(positive_data_file, negative_data_file):
    """
    Loads MR polarity data from files, splits the data into words and generates labels.
    Returns split sentences and labels.
    """
    # Load data from files
    positive_examples = list(open(positive_data_file, "r", encoding='utf-8').readlines())
    positive_examples = [s.strip() for s in positive_examples]
    negative_examples = list(open(negative_data_file, "r", encoding='utf-8').readlines())
    negative_examples = [s.strip() for s in negative_examples]
    # Split by words
    x = positive_examples + negative_examples
    x = [clean_str(sent) for sent in x]
    x = np.array(x)
    # Generate labels
    positive_labels = [1] * len(positive_examples)
    negative_labels = [0] * len(negative_examples)
    y = np.concatenate([positive_labels, negative_labels], 0)


    shuffle_indices = np.random.permutation(np.arange(len(y)))
    shuffled_x = x[shuffle_indices]
    shuffled_y = y[shuffle_indices]
```

```
        return shuffled_x, shuffled_y
```

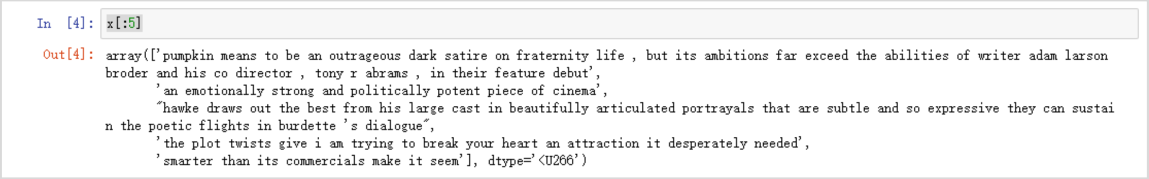
Load data:

```
positive_data_file = 'data/rt-polarity.pos'
negative_data_file = 'data/rt-polarity.neg'
x, y = load_data_and_labels(positive_data_file, negative_data_file)
```

Show data features:

```
x[:5]
```

Output:

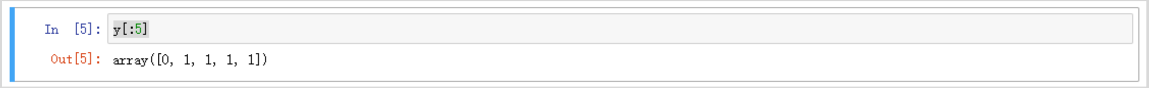


```
In  [4]: x[:5]
Out[4]: array(['pumpkin means to be an outrageous dark satire on fraternity life , but its ambitions far exceed the abilities of writer adam larson
        broder and his co director , tony r abrams , in their feature debut',
               'an emotionally strong and politically potent piece of cinema',
               "hawke draws out the best from his large cast in beautifully articulated portrayals that are subtle and so expressive they can sustai
        n the poetic flights in burdette 's dialogue",
               'the plot twists give i am trying to break your heart an attraction it desperately needed',
               'smarter than its commercials make it seem'], dtype='<U266')
```

Show data labels:

```
y[:5]
```

Output:



```
In  [5]: y[:5]
Out[5]: array([0, 1, 1, 1, 1])
```

Input:

```
test_size = 2000
x_train, y_train = x[:-2000], y[:-2000]
x_test, y_test = x[-2000:], y[-2000:]
label_map = {0: 'negative', 1: 'positive'}
```

Step 4 Define the main class of the classifier and define the training and test functions.

Input:

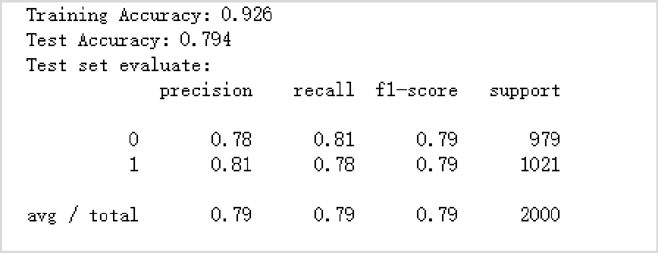
```
class NB_Classifier(object):

    def __init__(self):
        # naive bayes
        self.model = MultinomialNB( alpha=1) #Laplace smooth：1
        # use tf-idf extract features
        self.feature_processor = TfidfVectorizer()

    def fit(self, x_train, y_train, x_test, y_test):
        # tf-idf extract features
        x_train_fea = self.feature_processor.fit_transform(x_train)
        self.model.fit(x_train_fea, y_train)

        train_accuracy = self.model.score(x_train_fea, y_train)
        print("Training Accuracy： {}".format(round(train_accuracy, 3)))

        x_test_fea = self.feature_processor.transform(x_test)
        y_predict = self.model.predict(x_test_fea)
        test_accuracy = accuracy_score(y_test, y_predict)
        print("Test Accuracy： {}".format(round(test_accuracy, 3)))

        y_predict = self.model.predict(x_test_fea)
        print('Test set evaluate： ')
        print(classification_report(y_test, y_predict, target_names=['0', '1']))

    def single_predict(self, text):

        text_fea = self.feature_processor.transform([text])
        predict_idx = self.model.predict(text_fea)[0]
        predict_label = label_map[predict_idx]
        predict_prob = self.model.predict_proba(text_fea)[0][predict_idx]

        return predict_label, predict_prob
```

Step 5 Initialize and train the classifier.

Input:

```
nb_classifier = NB_Classifier()
nb_classifier.fit(x_train, y_train, x_test, y_test)
```

Output:



```
Training Accuracy: 0.926
Test Accuracy: 0.794
Test set evaluate:
             precision    recall  f1-score   support

          0       0.78      0.81      0.79       979
          1       0.81      0.78      0.79      1021

avg / total       0.79      0.79      0.79      2000
```

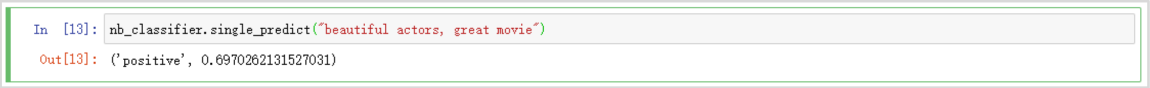
Step 6 Single sentence test

Test the prediction result of a single sentence:

Input:

```
nb_classifier.single_predict("beautiful actors, great movie")
```

Output:

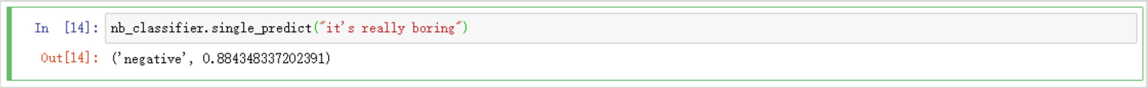


```
In  [13]: nb_classifier.single_predict("beautiful actors, great movie")
Out[13]: ('positive', 0.69702621315270031)
```

Input:

```
nb_classifier.single_predict("it's really boring")
```

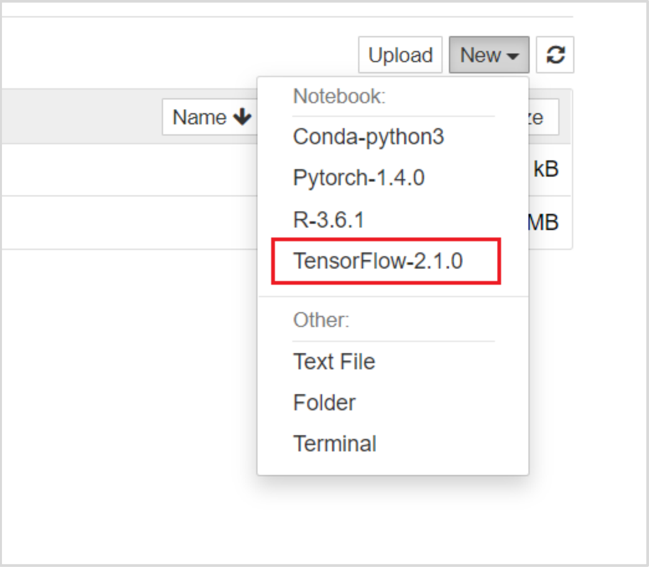
Output:



```
In  [14]: nb_classifier.single_predict("it's really boring")
Out[14]: ('negative', 0.884348337202391)
```

## 2.3.3 SVM Text Classification

Step 1 Create a notebook file and select the TensorFlow-2.1.0 environment.



Step 2 Importing Related Modules

```
import re
import pandas as pd
import numpy as np
from sklearn import svm
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.metrics import classification_report, accuracy_score
```

Step 3 Data preprocessing

Input:

```
def clean_str(string):
    """
    Tokenization/string cleaning for all datasets except for SST.
    Original taken from https://github.com/yoonkim/CNN_sentence/blob/master/process_data.py
    """
    string = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"\'s", " \'s", string)
    string = re.sub(r"\'ve", " \'ve", string)
    string = re.sub(r"n\'t", " n\'t", string)
    string = re.sub(r"\'re", " \'re", string)
    string = re.sub(r"\'d", " \'d", string)
    string = re.sub(r"\'ll", " \'ll", string)
    string = re.sub(r",", " , ", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", " \( ", string)
    string = re.sub(r"\)", " \) ", string)
    string = re.sub(r"\?", " \? ", string)
    string = re.sub(r"\s{2,}", " ", string)
    return string.strip().lower()


def load_data_and_labels(positive_data_file, negative_data_file):
    """
    Loads MR polarity data from files, splits the data into words and generates labels.
    Returns split sentences and labels.
    """
    # Load data from files
    positive_examples = list(open(positive_data_file, "r", encoding='utf-8').readlines())
    positive_examples = [s.strip() for s in positive_examples]
    negative_examples = list(open(negative_data_file, "r", encoding='utf-8').readlines())
    negative_examples = [s.strip() for s in negative_examples]
    # Split by words
    x = positive_examples + negative_examples
    x = [clean_str(sent) for sent in x]
    x = np.array(x)
    # Generate labels
    positive_labels = [1] * len(positive_examples)
    negative_labels = [0] * len(negative_examples)
    y = np.concatenate([positive_labels, negative_labels], 0)


    shuffle_indices = np.random.permutation(np.arange(len(y)))
    shuffled_x = x[shuffle_indices]
    shuffled_y = y[shuffle_indices]

    return shuffled_x, shuffled_y
```

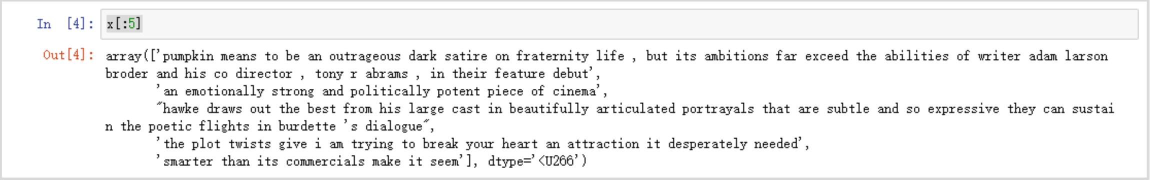
Load data:

```
positive_data_file = 'data/rt-polarity.pos'
negative_data_file = 'data/rt-polarity.neg'
x, y = load_data_and_labels(positive_data_file, negative_data_file)
```

Show data features:

```
x[:5]
```

Output:

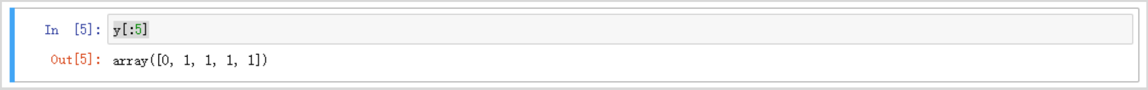


```
In  [4]: x[:5]

Out[4]: array(['pumpkin means to be an outrageous dark satire on fraternity life , but its ambitions far exceed the abilities of writer adam larson
        broder and his co director , tony r abrams , in their feature debut',
               'an emotionally strong and politically potent piece of cinema',
               "hawke draws out the best from his large cast in beautifully articulated portrayals that are subtle and so expressive they can sustai
        n the poetic flights in burdette 's dialogue",
               'the plot twists give i am trying to break your heart an attraction it desperately needed',
               'smarter than its commercials make it seem'], dtype='<U266')
```

Show data labels:

```
y[:5]
```

Output:



```
In  [5]: y[:5]
Out[5]: array([0, 1, 1, 1, 1])
```

Input:

```
test_size = 2000
x_train, y_train = x[:-2000], y[:-2000]
x_test, y_test = x[-2000:], y[-2000:]
label_map = {0: 'negative', 1: 'positive'}
```

Step 4 Define the main class of the classifier, define training, and test functions.

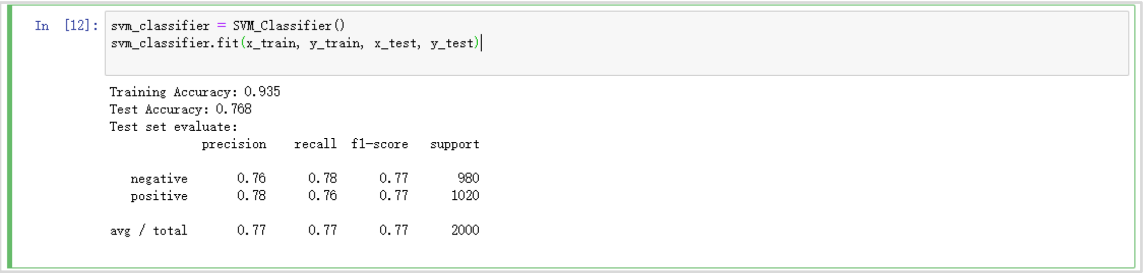
```
class SVM_Classifier(object):

    def __init__(self, use_chi=False):

        self.use_chi = use_chi # Whether use chi-square test for feature selection
        # SVM
        self.model = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
        # use tf-idf extract features
        self.feature_processor = TfidfVectorizer()
        # chi-square test for feature selection
        if use_chi:
            self.feature_selector = SelectKBest(chi2, k=10000) # 34814 -> 10000

    def fit(self, x_train, y_train, x_test, y_test):

        x_train_fea = self.feature_processor.fit_transform(x_train)
        if self.use_chi:
            x_train_fea = self.feature_selector.fit_transform(x_train_fea, y_train)
        self.model.fit(x_train_fea, y_train)

        train_accuracy = self.model.score(x_train_fea, y_train)
        print("Training Accuracy：{}".format(round(train_accuracy, 3)))

        x_test_fea = self.feature_processor.transform(x_test)
        if self.use_chi:
            x_test_fea = self.feature_selector.transform(x_test_fea)
        y_predict = self.model.predict(x_test_fea)
        test_accuracy = accuracy_score(y_test, y_predict)
        print("Test Accuracy：{}".format(round(test_accuracy, 3)))
        print('Test set evaluate：')
        print(classification_report(y_test, y_predict, target_names=['negative', 'positive']))

    def single_predict(self, text):
        text_fea = self.feature_processor.transform([text])
        if self.use_chi:
            text_fea = self.feature_selector.transform(text_fea)
        predict_idx = self.model.predict(text_fea)[0]
        predict_label = label_map[predict_idx]
        return predict_label
```

## Step 5 Train the SVM classifier without the chi-square test.

Input:

```
svm_classifier = SVM_Classifier()
svm_classifier.fit(x_train, y_train, x_test, y_test)
```

Output:



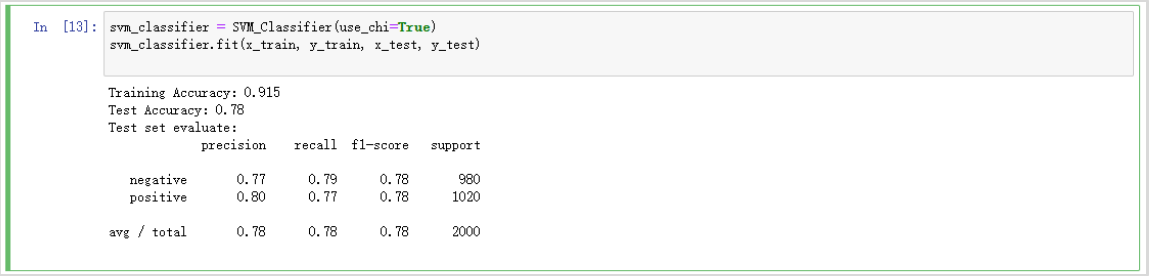
```
In [12]: svm_classifier = SVM_Classifier()
         svm_classifier.fit(x_train, y_train, x_test, y_test)

Training Accuracy: 0.935
Test Accuracy: 0.768
Test set evaluate:
             precision    recall  f1-score   support

   negative       0.76      0.78      0.77       980
   positive       0.78      0.76      0.77      1020

avg / total       0.77      0.77      0.77      2000
```

## Step 6 Train SVM classifiers and use chi-square test.

Input:

```
svm_classifier = SVM_Classifier(use_chi=True)
svm_classifier.fit(x_train, y_train, x_test, y_test)
```

Output:



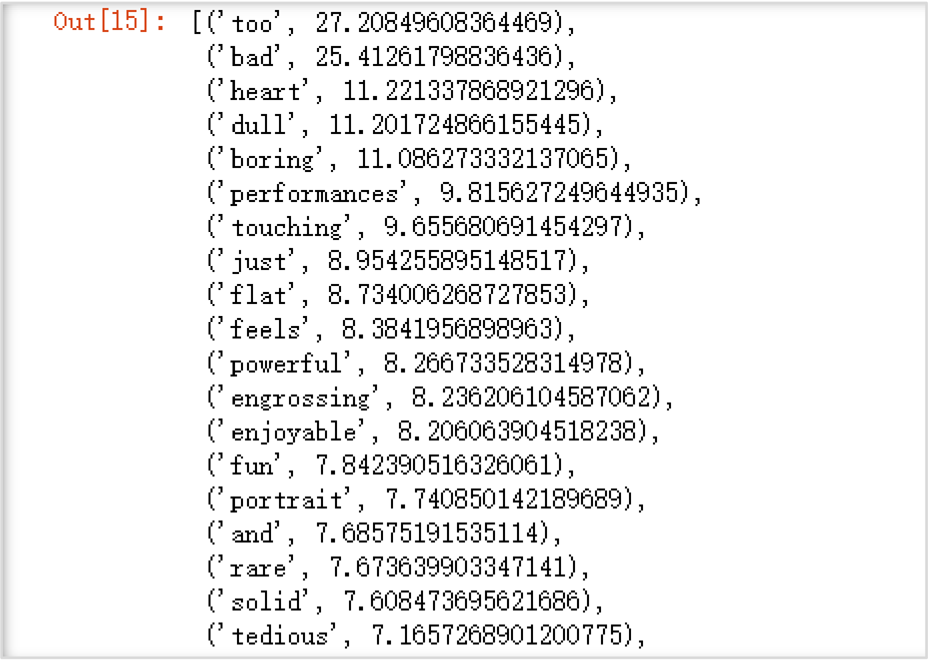
```
In [13]: svm_classifier = SVM_Classifier(use_chi=True)
         svm_classifier.fit(x_train, y_train, x_test, y_test)

Training Accuracy: 0.915
Test Accuracy: 0.78
Test set evaluate:
             precision    recall  f1-score   support

   negative       0.77      0.79      0.78       980
   positive       0.80      0.77      0.78      1020

avg / total       0.78      0.78      0.78      2000
```

## Step 7 chi-square feature analysis

Input:

```
def feature_analysis():
    feature_names = svm_classifier.feature_processor.get_feature_names()
    feature_scores = svm_classifier.feature_selector.scores_
    fea_score_tups = list(zip(feature_names, feature_scores))
    fea_score_tups.sort(key=lambda tup: tup[1], reverse=True)

    return fea_score_tups
feature_analysis()[:500]
```

Output:



```
Out[15]: [('too', 27.20849608364469),
          ('bad', 25.41261798836436),
          ('heart', 11.221337868921296),
          ('dull', 11.201724866155445),
          ('boring', 11.086273332137065),
          ('performances', 9.815627249644935),
          ('touching', 9.655680691454297),
          ('just', 8.954255895148517),
          ('flat', 8.734006268727853),
          ('feels', 8.3841956898963),
          ('powerful', 8.266733528314978),
          ('engrossing', 8.236206104587062),
          ('enjoyable', 8.206063904518238),
          ('fun', 7.842390516326061),
          ('portrait', 7.740850142189689),
          ('and', 7.68575191535114),
          ('rare', 7.673639903347141),
          ('solid', 7.608473695621686),
          ('tedious', 7.1657268901200775),
```

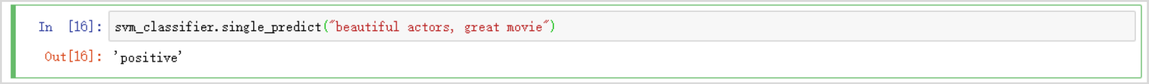
Step 8 Single sentence test

Test the prediction result of a single sentence:

Input:

```
svm_classifier.single_predict("beautiful actors, great movie")
```

Output:

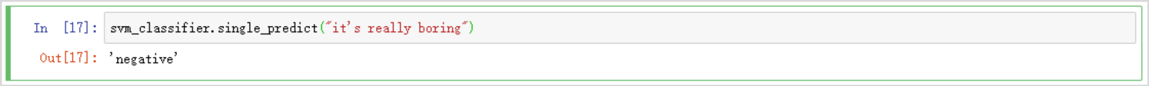


```
In  [16]: svm_classifier.single_predict("beautiful actors, great movie")
Out[16]: 'positive'
```

Input:

```
svm_classifier.single_predict("it's really boring")
```

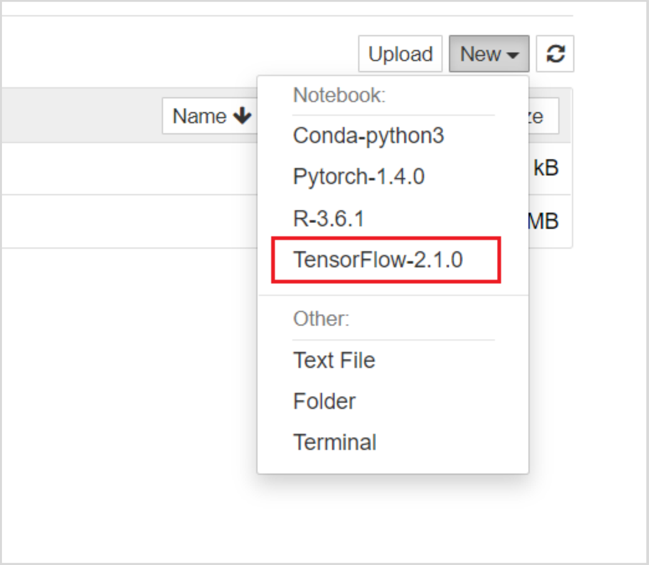
Output:



```
In  [17]: svm_classifier.single_predict("it's really boring")
Out[17]: 'negative'
```

## 2.3.4 TextCNN Text Classification

Step 1 Create a notebook file and select the TensorFlow-2.1.0 environment.



## Step 2 Importing Related Library

```
import re
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.metrics import classification_report
```

## Step 3 Data preprocessing

Input:

```
def clean_str(string):
    """
    Tokenization/string cleaning for all datasets except for SST.
    Original taken from https://github.com/yoonkim/CNN_sentence/blob/master/process_data.py
    """
    string = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"\'s", " \'s", string)
    string = re.sub(r"\'ve", " \'ve", string)
    string = re.sub(r"n\'t", " n\'t", string)
    string = re.sub(r"\'re", " \'re", string)
    string = re.sub(r"\'d", " \'d", string)
    string = re.sub(r"\'ll", " \'ll", string)
    string = re.sub(r",", " , ", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", " \( ", string)
    string = re.sub(r"\)", " \) ", string)
    string = re.sub(r"\?", " \? ", string)
    string = re.sub(r"\s{2,}", " ", string)
    return string.strip().lower()


def load_data_and_labels(positive_data_file, negative_data_file):
    """
    Loads MR polarity data from files, splits the data into words and generates labels.
    Returns split sentences and labels.
    """
    # Load data from files
    positive_examples = list(open(positive_data_file, "r", encoding='utf-8').readlines())
    positive_examples = [s.strip() for s in positive_examples]
    negative_examples = list(open(negative_data_file, "r", encoding='utf-8').readlines())
    negative_examples = [s.strip() for s in negative_examples]
    # Split by words
    x = positive_examples + negative_examples
    x = [clean_str(sent) for sent in x]
    x = np.array(x)
    # Generate labels
    positive_labels = [1] * len(positive_examples)
    negative_labels = [0] * len(negative_examples)
    y = np.concatenate([positive_labels, negative_labels], 0)


    shuffle_indices = np.random.permutation(np.arange(len(y)))
    shuffled_x = x[shuffle_indices]
    shuffled_y = y[shuffle_indices]

    return shuffled_x, shuffled_y
```

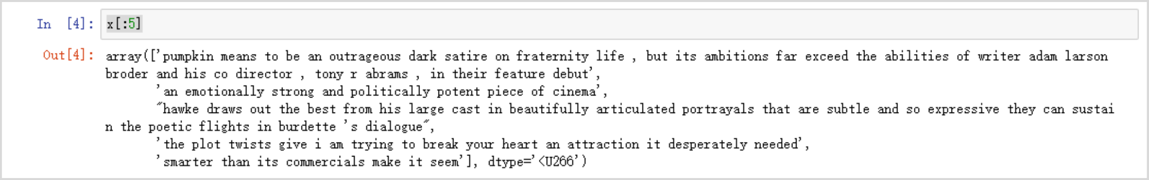
Load data:

```
positive_data_file = 'data/rt-polarity.pos'
negative_data_file = 'data/rt-polarity.neg'
x, y = load_data_and_labels(positive_data_file, negative_data_file)
```

Show data features:

```
x[:5]
```

Output:

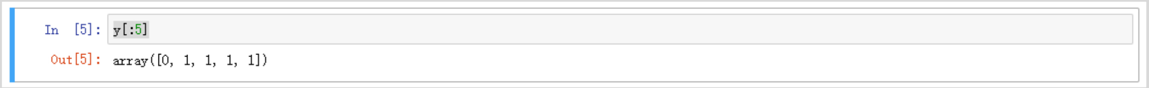


```
In  [4]: x[:5]

Out[4]: array(['pumpkin means to be an outrageous dark satire on fraternity life , but its ambitions far exceed the abilities of writer adam larson
        broder and his co director , tony r abrams , in their feature debut',
               'an emotionally strong and politically potent piece of cinema',
               "hawke draws out the best from his large cast in beautifully articulated portrayals that are subtle and so expressive they can sustai
        n the poetic flights in burdette 's dialogue",
               'the plot twists give i am trying to break your heart an attraction it desperately needed',
               'smarter than its commercials make it seem'], dtype='<U266')
```

Show data labels:

```
y[:5]
```

Output:

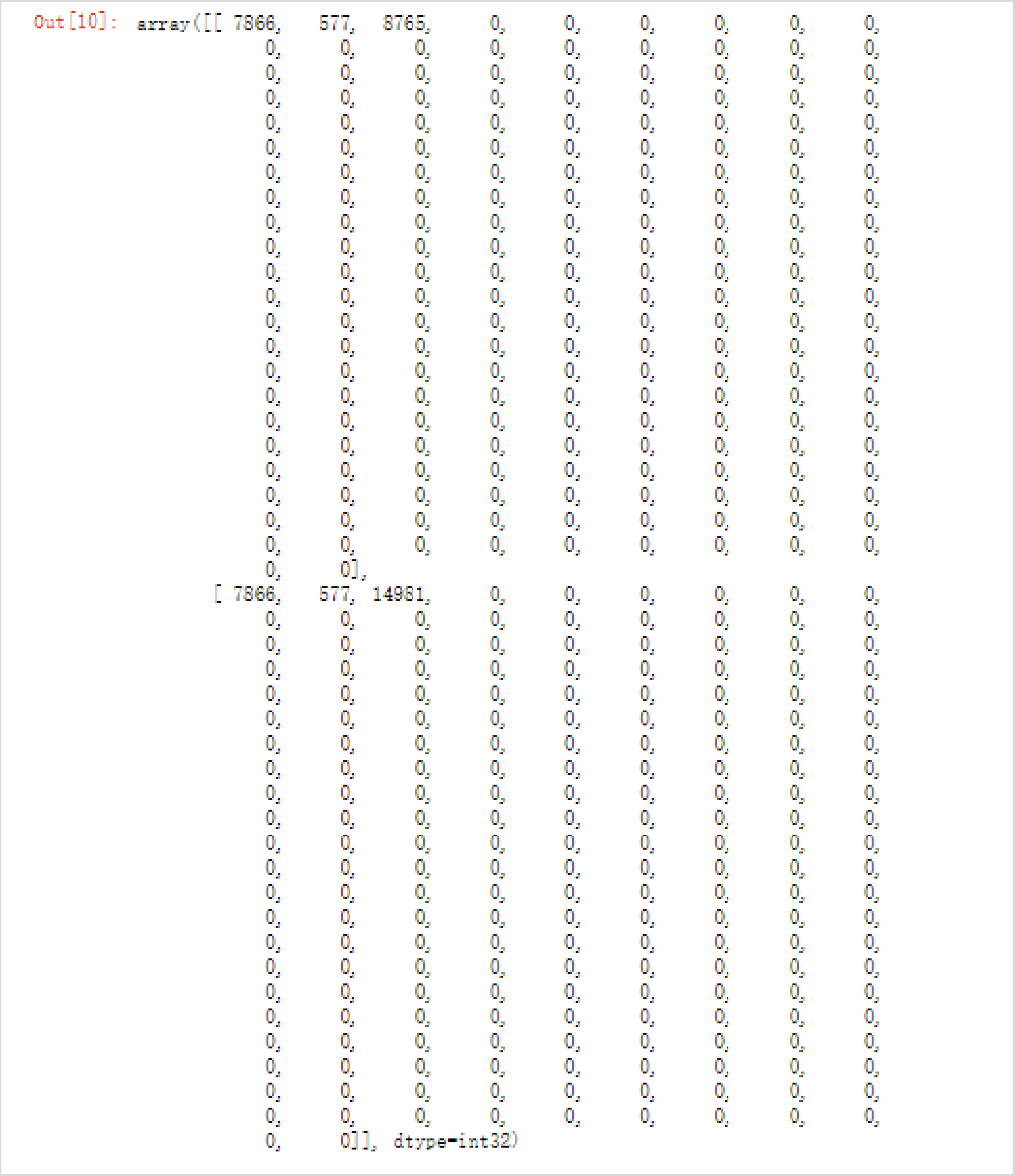


```
In  [5]: y[:5]

Out[5]: array([0, 1, 1, 1, 1])
```

Input:

```
vocab = set()
for doc in x:
    for word in doc.split(' '):
        if word.strip():
            vocab.add(word.strip().lower())

# write to vocab.txt file
with open('data/vocab.txt', 'w') as file:
    for word in   vocab:
        file.write(word)
        file.write('\n')
test_size = 2000
x_train, y_train = x[:-2000], y[:-2000]
x_test, y_test = x[-2000:], y[-2000:]
label_map = {0: 'negative', 1: 'positive'}

class Config():
    embedding_dim = 100 # word embedding dimention
    max_seq_len = 200 # max sequence length
    vocab_file = 'data/vocab.txt' # vocab_file_length
config = Config()

class Preprocessor():
    def __init__(self, config):
        self.config = config
        # initial the map of word and index
        token2idx = {"[PAD]": 0, "[UNK]": 1} # {word：id}
        with open(config.vocab_file, 'r') as reader:
            for index, line in enumerate(reader):
                token = line.strip()
                token2idx[token] = index+2

        self.token2idx = token2idx

    def transform(self, text_list):
        # tokenization, and transform word to coresponding index
        idx_list = [[self.token2idx.get(word.strip().lower(), self.token2idx['[UNK]']) for word in
text.split(' ')] for text in text_list]
        idx_padding = pad_sequences(idx_list, self.config.max_seq_len, padding='post')

        return idx_padding

preprocessor = Preprocessor(config)
preprocessor.transform(['I love working', 'I love eating'])
```

Output:



```
Out[10]: array([[ 7866,   577,  8765,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0],
                [ 7866,   577, 14981,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0,     0,     0,     0,     0,     0,     0,     0,
                      0,     0]], dtype=int32)
```

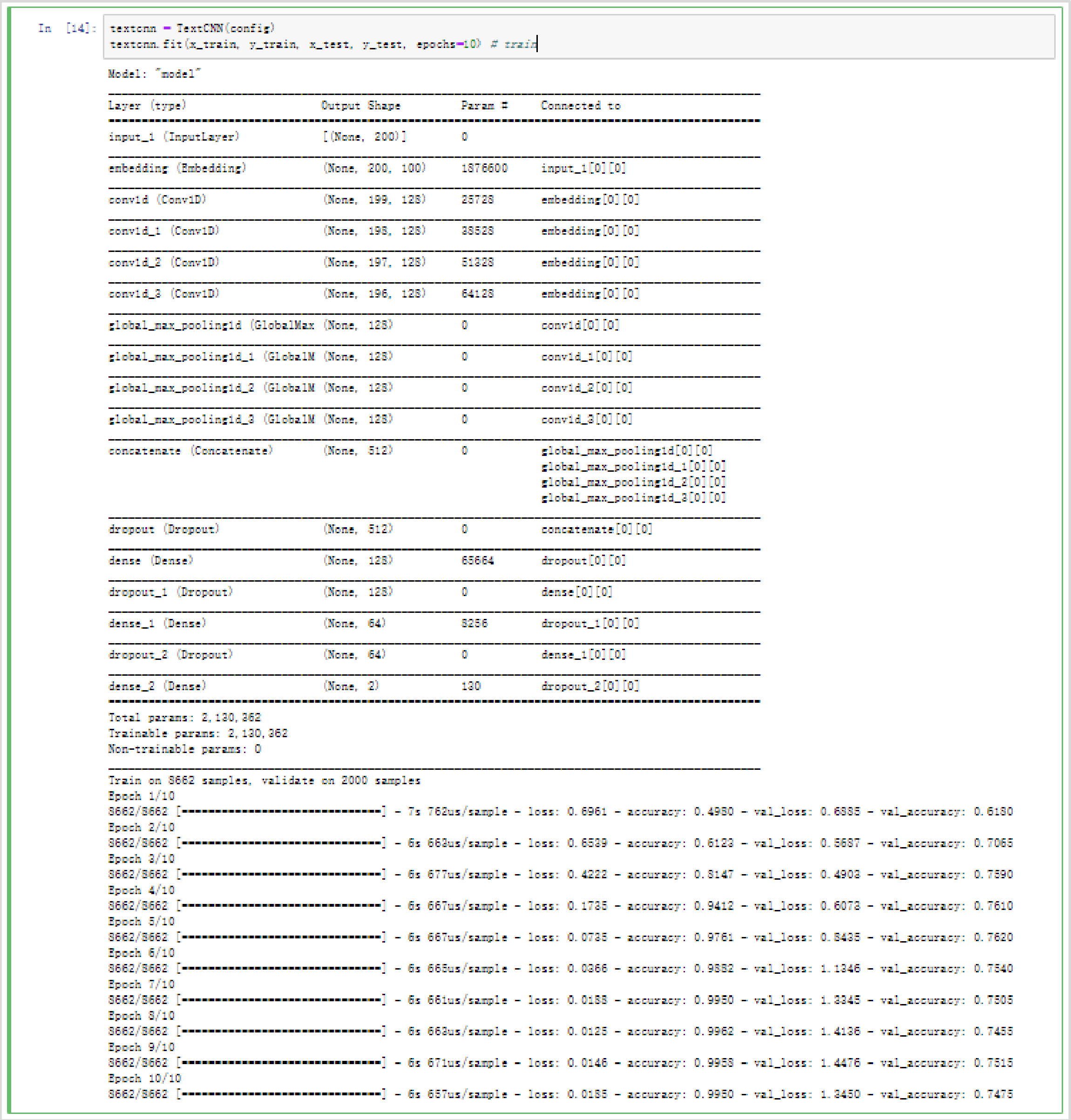
Step 4 Defines the TextCNN main class, including model building, training, and test functions.

```
class TextCNN(object):
    def __init__(self, config):
        self.config = config
        self.preprocessor = Preprocessor(config)
        self.class_name = {0: 'negative', 1: 'positive'}

    def build_model(self):
        # build model architecture
        idx_input = tf.keras.layers.Input((self.config.max_seq_len,))
        input_embedding = tf.keras.layers.Embedding(len(self.preprocessor.token2idx),
                    self.config.embedding_dim,
                    input_length=self.config.max_seq_len,
                    mask_zero=True)(idx_input)
        convs = []
        for kernel_size in [2, 3, 4, 5]:
            c = tf.keras.layers.Conv1D(128, kernel_size, activation='relu')(input_embedding)
            c = tf.keras.layers.GlobalMaxPooling1D()(c)
            convs.append(c)
        fea_cnn = tf.keras.layers.Concatenate()(convs)
        fea_cnn = tf.keras.layers.Dropout(rate=0.5)(fea_cnn)
        fea_dense = tf.keras.layers.Dense(128, activation='relu')(fea_cnn)
        fea_dense = tf.keras.layers.Dropout(rate=0.5)(fea_dense)
        fea_dense = tf.keras.layers.Dense(64, activation='relu')(fea_dense)
        fea_dense = tf.keras.layers.Dropout(rate=0.3)(fea_dense)
        output = tf.keras.layers.Dense(2, activation='softmax')(fea_dense)

        model = tf.keras.Model(inputs=idx_input, outputs=output)
        model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

        model.summary()

        self.model = model

    def fit(self, x_train, y_train, x_valid=None, y_valid=None, epochs=5, batch_size=128, **kwargs):
        # train
        self.build_model()

        x_train = self.preprocessor.transform(x_train)
        if x_valid is not None and y_valid is not None:
            x_valid = self.preprocessor.transform(x_valid)

        self.model.fit(
            x=x_train,
            y=y_train,
            validation_data= (x_valid, y_valid) if x_valid is not None and y_valid is not None else
None,
            batch_size=batch_size,
            epochs=epochs,
            **kwargs
        )

    def evaluate(self, x_test, y_test):
        # evaluate
        x_test = self.preprocessor.transform(x_test)
        y_pred_probs = self.model.predict(x_test)
        y_pred = np.argmax(y_pred_probs, axis=-1)
        result = classification_report(y_test, y_pred, target_names=['negative', 'positive'])
        print(result)


    def single_predict(self, text):
        # predict
        input_idx = self.preprocessor.transform([text])
        predict_prob = self.model.predict(input_idx)[0]
        predict_label_id = np.argmax(predict_prob)
        predict_label_name = self.class_name[predict_label_id]
        predict_label_prob = predict_prob[predict_label_id]

        return predict_label_name, predict_label_prob
```

## Step 5 Initialize the model and train the model.

```
textcnn = TextCNN(config)
textcnn.fit(x_train, y_train, x_test, y_test, epochs=10) # train
```

Output:

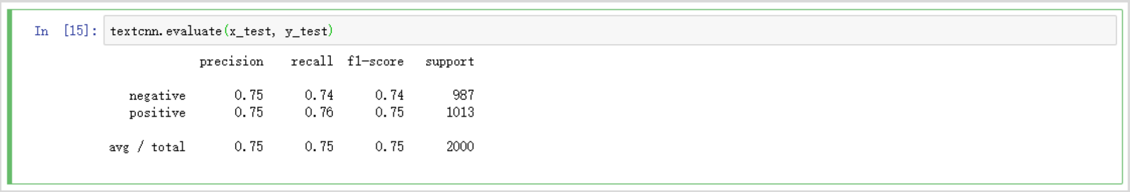


```
In [14]: textcnn = TextCNN(config)
         textcnn.fit(x_train, y_train, x_test, y_test, epochs=10) # train

Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 200)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 200, 100)     1876600     input_1[0][0]
__________________________________________________________________________________________________
conv1d (Conv1D)                 (None, 199, 128)     25728       embedding[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 198, 128)     38528       embedding[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 197, 128)     51328       embedding[0][0]
__________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 196, 128)     64128       embedding[0][0]
__________________________________________________________________________________________________
global_max_pooling1d (GlobalMax (None, 128)          0           conv1d[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM (None, 128)          0           conv1d_1[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_2 (GlobalM (None, 128)          0           conv1d_2[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_3 (GlobalM (None, 128)          0           conv1d_3[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 512)          0           global_max_pooling1d[0][0]
                                                                 global_max_pooling1d_1[0][0]
                                                                 global_max_pooling1d_2[0][0]
                                                                 global_max_pooling1d_3[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 512)          0           concatenate[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 128)          65664       dropout[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 128)          0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 64)           8256        dropout_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 64)           0           dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 2)            130         dropout_2[0][0]
==================================================================================================
Total params: 2,130,362
Trainable params: 2,130,362
Non-trainable params: 0
__________________________________________________________________________________________________
Train on 8662 samples, validate on 2000 samples
Epoch 1/10
8662/8662 [==============================] - 7s 762us/sample - loss: 0.6961 - accuracy: 0.4980 - val_loss: 0.6835 - val_accuracy: 0.6180
Epoch 2/10
8662/8662 [==============================] - 6s 663us/sample - loss: 0.6539 - accuracy: 0.6123 - val_loss: 0.5637 - val_accuracy: 0.7065
Epoch 3/10
8662/8662 [==============================] - 6s 677us/sample - loss: 0.4222 - accuracy: 0.8147 - val_loss: 0.4903 - val_accuracy: 0.7590
Epoch 4/10
8662/8662 [==============================] - 6s 667us/sample - loss: 0.1735 - accuracy: 0.9412 - val_loss: 0.6073 - val_accuracy: 0.7610
Epoch 5/10
8662/8662 [==============================] - 6s 667us/sample - loss: 0.0735 - accuracy: 0.9761 - val_loss: 0.8435 - val_accuracy: 0.7620
Epoch 6/10
8662/8662 [==============================] - 6s 665us/sample - loss: 0.0366 - accuracy: 0.9882 - val_loss: 1.1346 - val_accuracy: 0.7540
Epoch 7/10
8662/8662 [==============================] - 6s 661us/sample - loss: 0.0188 - accuracy: 0.9950 - val_loss: 1.3345 - val_accuracy: 0.7505
Epoch 8/10
8662/8662 [==============================] - 6s 663us/sample - loss: 0.0125 - accuracy: 0.9962 - val_loss: 1.4136 - val_accuracy: 0.7455
Epoch 9/10
8662/8662 [==============================] - 6s 671us/sample - loss: 0.0146 - accuracy: 0.9958 - val_loss: 1.4476 - val_accuracy: 0.7515
Epoch 10/10
8662/8662 [==============================] - 6s 657us/sample - loss: 0.0185 - accuracy: 0.9950 - val_loss: 1.3450 - val_accuracy: 0.7475
```

## Step 6 Test Set Evaluation

```
textcnn.evaluate(x_test, y_test) # Test Set Evaluation
```

Output:



```
In [15]: textcnn.evaluate(x_test, y_test)
             precision    recall  f1-score   support

   negative       0.75      0.74      0.74       987
   positive       0.75      0.76      0.75      1013

avg / total       0.75      0.75      0.75      2000
```

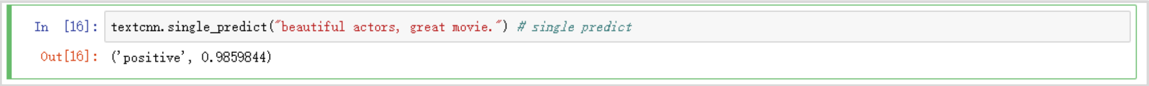
Step 7 Single sentence test

Test the prediction result of a single sentence:

Input:

```
textcnn.single_predict("beautiful actors, great movie.") # single sentence predict
```

Output:

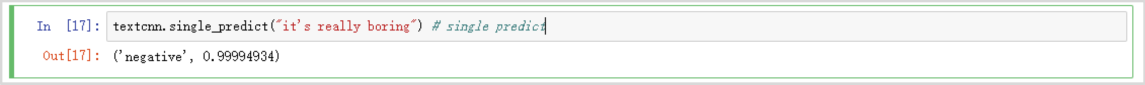


```
In [16]: textcnn.single_predict("beautiful actors, great movie.") # single predict
Out[16]: ('positive', 0.9859844)
```

Input:

```
textcnn.single_predict("it's really boring") # single sentence predict
```

Output:



```
In [17]: textcnn.single_predict("it's really boring") # single predict
Out[17]: ('negative', 0.99994934)
```

## 2.4 Experiment Summary

This chapter introduces the implementation of text classification tasks in NLP, through an application case of sentiment analysis. And this chapter compares the differences between three algorithms: Naive Bayes, SVM, and TextCNN. Through experiments, trainees can understand text classification tasks and Naive Bayes, SVM and TextCNN algorithms deeply.

# 3 Machine Translation

## 3.1 Introduction

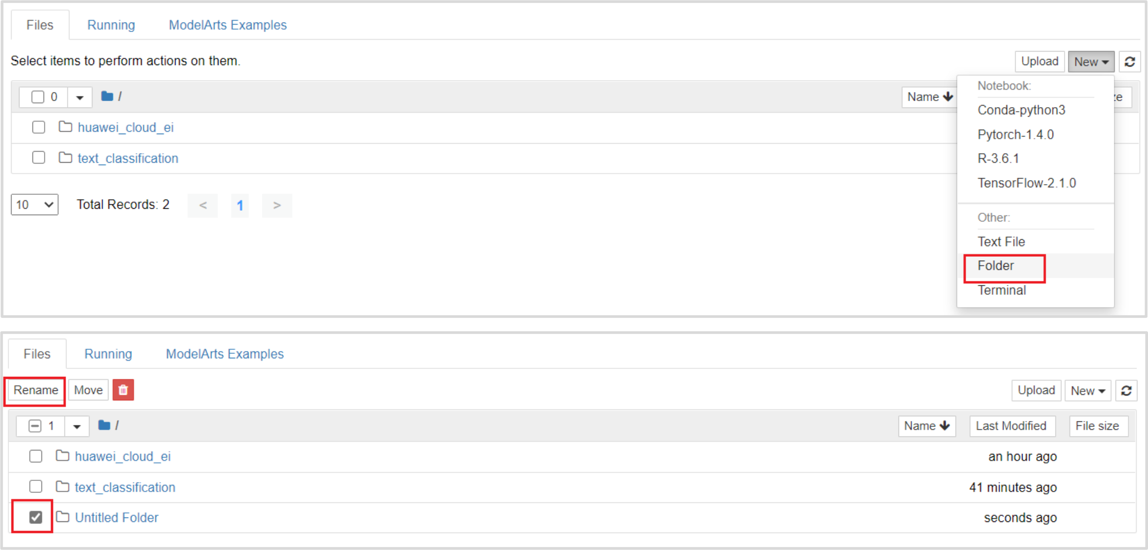
This experiment describes how to use TensorFlow to build a machine translation model based on the “encoder-decoder” architecture and use the “attention” mechanism to further enhance the effect.

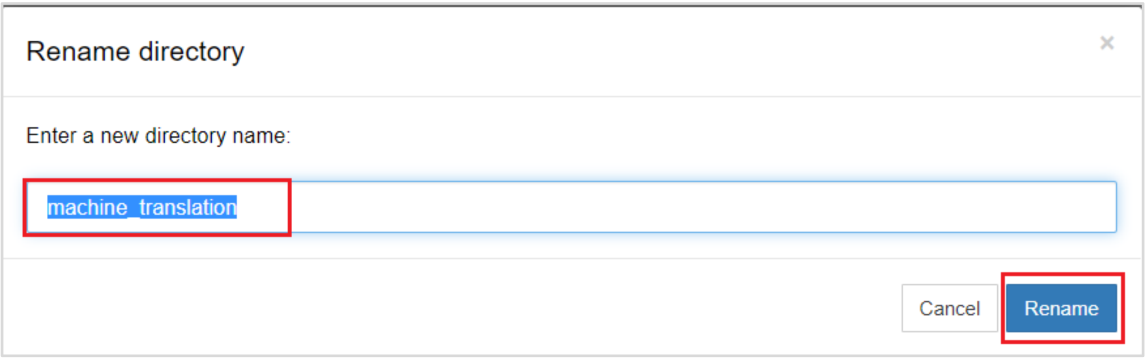
## 3.2 Objective

- Understand the basic principles of the encoder-decoder architecture.
- Understand the algorithm process of machine translation.
- Master the method of building a machine translation model using TensorFlow.

## 3.3 Procedure

Step 1 Go to the notebook home page, create a folder, and rename the folder machine_translation.



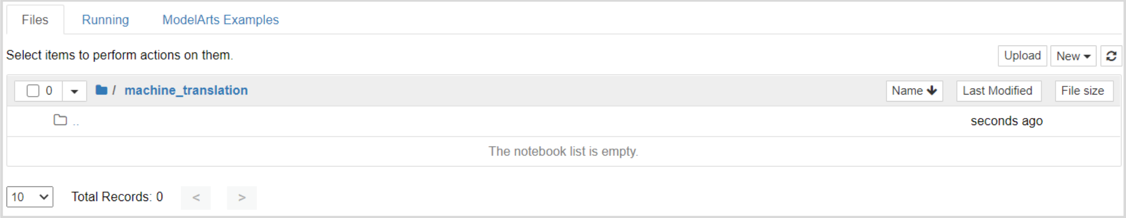


Rename directory

Enter a new directory name:

machine_translation

Step 2 Click the created machine_translation folder.



Select items to perform actions on them.

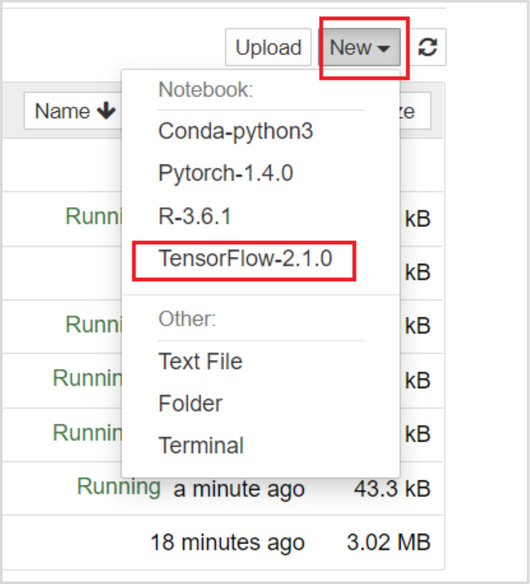
machine_translation

seconds ago

The notebook list is empty.

Total Records: 0

Step 3 Create a notebook file and select the TensorFlow-2.1.0 environment.

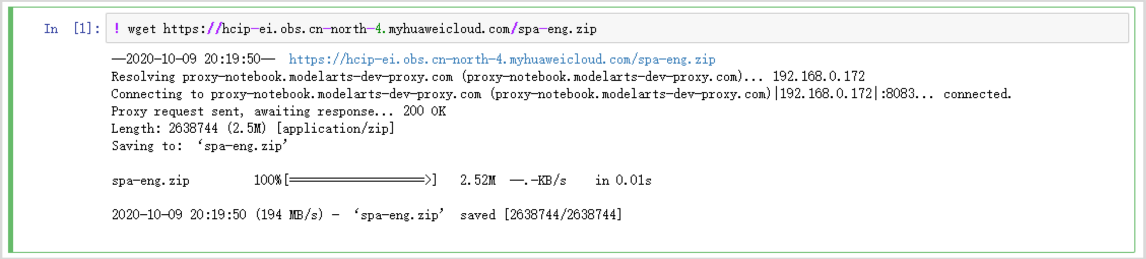


Step 4 Downloading Data

Input:

```
! wget https://huawei-ai-certification.obs.cn-north-4.myhuaweicloud.com/ENG/HCIP-AI%20EI%20Developer/V2.1/spa-eng.zip
```

Output:



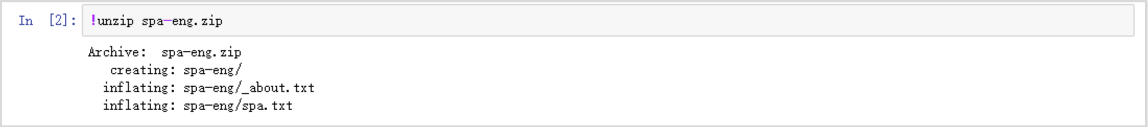
```
In  [1]: ! wget https://hcip-ei.obs.cn-north-4.myhuaweicloud.com/spa-eng.zip

         —2020-10-09 20:19:50—  https://hcip-ei.obs.cn-north-4.myhuaweicloud.com/spa-eng.zip
         Resolving proxy-notebook.modelarts-dev-proxy.com (proxy-notebook.modelarts-dev-proxy.com)... 192.168.0.172
         Connecting to proxy-notebook.modelarts-dev-proxy.com (proxy-notebook.modelarts-dev-proxy.com)|192.168.0.172|:8083... connected.
         Proxy request sent, awaiting response... 200 OK
         Length: 2638744 (2.5M) [application/zip]
         Saving to: ‘spa-eng.zip’

         spa-eng.zip         100%[==================>]   2.52M  —.-KB/s    in 0.01s

         2020-10-09 20:19:50 (194 MB/s) - ‘spa-eng.zip’ saved [2638744/2638744]
```

## Step 5 Decompressing Data

Input:

```
!unzip spa-eng.zip
```

Output:



```
In  [2]: !unzip spa-eng.zip

         Archive:  spa-eng.zip
            creating: spa-eng/
           inflating: spa-eng/_about.txt
           inflating: spa-eng/spa.txt
```

## Step 6 Importing Related Library

Input:

```
import tensorflow as tf

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from sklearn.model_selection import train_test_split

import unicodedata
import re
import numpy as np
import os
import io
import time
```

## Step 7 Specifying the data path

Input:

```
path_to_file = "./spa-eng/spa.txt" ## dataset file
```

## Step 8 Defining a Preprocessing Function

Preprocessing includes:

- Converts the unicode file to ascii
- Replace particular characters with space
- Add a start and end token to the sentence

Input:

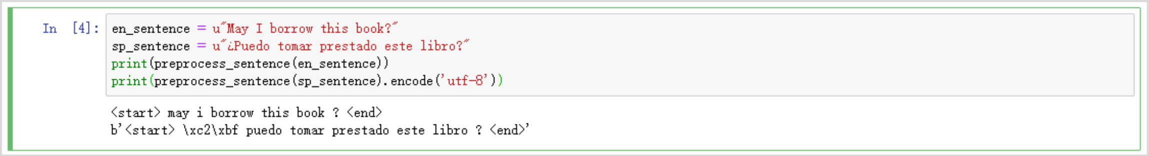
```
# Converts the unicode file to ascii
def unicode_to_ascii(s):
    return ''.join(c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn')


def preprocess_sentence(w):
    w = unicode_to_ascii(w.lower().strip())

    # creating a space between a word and the punctuation following it
    # eg: "he is a boy." => "he is a boy ."
    # Reference:- https://stackoverflow.com/questions/3645931/python-padding-punctuation-with-
white-spaces-keeping-punctuation
    w = re.sub(r"([?.!,¿])", r" \1 ", w)
    w = re.sub(r'[" "]+', " ", w)

    # replacing everything with space except (a-z, A-Z, ".", "?", "!", ",")
    w = re.sub(r"[^a-zA-Z?.!,¿]+", " ", w)

    w = w.strip()

    # adding a start and an end token to the sentence
    # so that the model know when to start and stop predicting.
    w = '<start> ' + w + ' <end>'
    return w
```

Preprocessing test:

Input:

```
en_sentence = u"May I borrow this book?"
sp_sentence = u"¿Puedo tomar prestado este libro?"
print(preprocess_sentence(en_sentence))
print(preprocess_sentence(sp_sentence).encode('utf-8'))
```

Output:



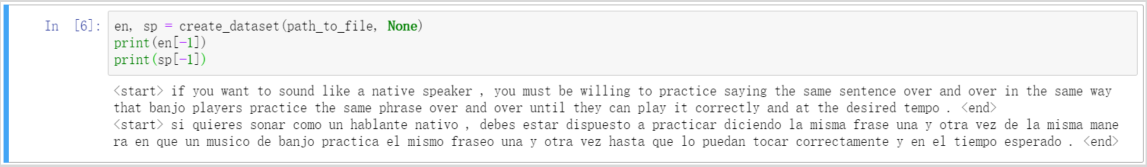
```
In  [4]: en_sentence = u"May I borrow this book?"
         sp_sentence = u"¿Puedo tomar prestado este libro?"
         print(preprocess_sentence(en_sentence))
         print(preprocess_sentence(sp_sentence).encode('utf-8'))

         <start> may i borrow this book ? <end>
         b'<start> \xc2\xbf puedo tomar prestado este libro ? <end>'
```

Input:

```
# 1. Remove the accents
# 2. Clean the sentences
# 3. Return word pairs in the format: [ENGLISH, SPANISH]
def create_dataset(path, num_examples):
    lines = io.open(path, encoding='UTF-8').read().strip().split('\n')

    word_pairs = [[preprocess_sentence(w) for w in l.split('\t')]   for l in lines[:num_examples]]

    return zip(*word_pairs)
```

```
en, sp = create_dataset(path_to_file, None)
print(en[-1])
print(sp[-1])
```

Output:



```
In  [6]: en, sp = create_dataset(path_to_file, None)
         print(en[-1])
         print(sp[-1])

         <start> if you want to sound like a native speaker , you must be willing to practice saying the same sentence over and over in the same way
         that banjo players practice the same phrase over and over until they can play it correctly and at the desired tempo . <end>
         <start> si quieres sonar como un hablante nativo , debes estar dispuesto a practicar diciendo la misma frase una y otra vez de la misma mane
         ra en que un musico de banjo practica el mismo fraseo una y otra vez hasta que lo puedan tocar correctamente y en el tiempo esperado . <end>
```

## Step 9 Load dataset

The operations include:

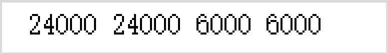
- Load the original data set.
- Preprocessing
- Convert text to ID

Input:

```
def tokenize(lang):
    lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(
        filters='')
    lang_tokenizer.fit_on_texts(lang)

    tensor = lang_tokenizer.texts_to_sequences(lang)

    tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
                                                           padding='post')

    return tensor, lang_tokenizer

def load_dataset(path, num_examples=None):
    # creating cleaned input, output pairs
    targ_lang, inp_lang = create_dataset(path, num_examples)

    input_tensor, inp_lang_tokenizer = tokenize(inp_lang)
    target_tensor, targ_lang_tokenizer = tokenize(targ_lang)

    return input_tensor, target_tensor, inp_lang_tokenizer, targ_lang_tokenizer

# Try experimenting with the size of that dataset
num_examples = 30000
input_tensor, target_tensor, inp_lang, targ_lang = load_dataset(path_to_file, num_examples)

# Calculate max_length of the target tensors
max_length_targ, max_length_inp = target_tensor.shape[1], input_tensor.shape[1]

# Creating training and validation sets using an 80-20 split
input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val =
train_test_split(input_tensor, target_tensor, test_size=0.2)

# Show length
```

```
print(len(input_tensor_train), len(target_tensor_train), len(input_tensor_val), len(target_tensor_val))
```

Output:



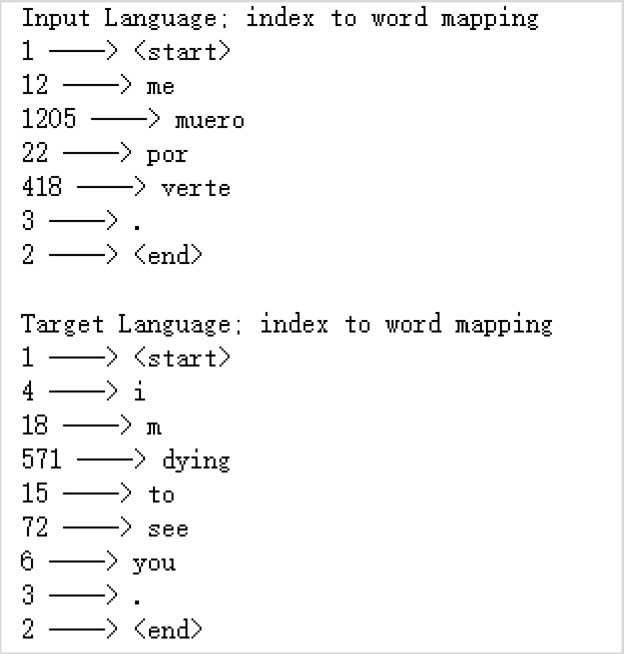
```
24000 24000 6000 6000
```

Convert text to ID:

Input:

```
def convert(lang, tensor):
    for t in tensor:
        if t!=0:
            print ("%d ----> %s" % (t, lang.index_word[t]))

print ("Input Language; index to word mapping")
convert(inp_lang, input_tensor_train[0])
print ()
print ("Target Language; index to word mapping")
convert(targ_lang, target_tensor_train[0])
```

Output:



```
Input Language; index to word mapping
1 ----> <start>
12 ----> me
1205 ----> muero
22 ----> por
418 ----> verte
3 ----> .
2 ----> <end>

Target Language; index to word mapping
1 ----> <start>
4 ----> i
18 ----> m
571 ----> dying
15 ----> to
72 ----> see
6 ----> you
3 ----> .
2 ----> <end>
```

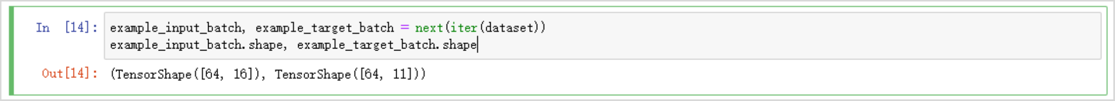
Step 10 Convert the file to tf.data.Dataset.

Input:

```
BUFFER_SIZE = len(input_tensor_train)
BATCH_SIZE = 64
steps_per_epoch = len(input_tensor_train)//BATCH_SIZE
embedding_dim = 256
units = 1024
vocab_inp_size = len(inp_lang.word_index)+1
```

```
vocab_tar_size = len(targ_lang.word_index)+1

dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train,
target_tensor_train)).shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)

example_input_batch, example_target_batch = next(iter(dataset))
example_input_batch.shape, example_target_batch.shape
```

Output:



```
In  [14]: example_input_batch, example_target_batch = next(iter(dataset))
          example_input_batch.shape, example_target_batch.shape

Out[14]: (TensorShape([64, 16]), TensorShape([64, 11]))
```

## Step 11 Defining an Encoder

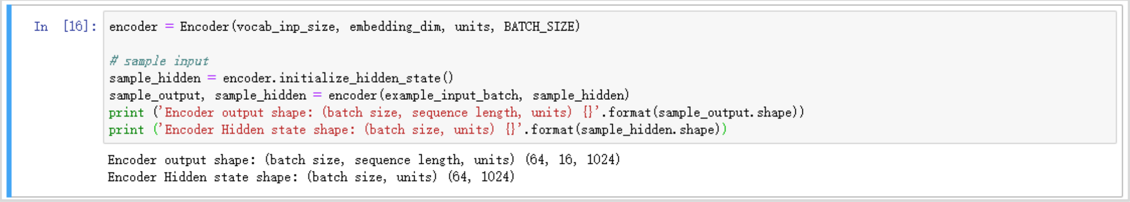
Input:

```
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.enc_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')

    def call(self, x, hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state = hidden)
        return output, state

    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.enc_units))
```

Input:

```
encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)

# sample input
sample_hidden = encoder.initialize_hidden_state()
sample_output, sample_hidden = encoder(example_input_batch, sample_hidden)
print ('Encoder output shape: (batch size, sequence length, units) {}'.format(sample_output.shape))
print ('Encoder Hidden state shape: (batch size, units) {}'.format(sample_hidden.shape))
```

Output:



```
In [16]: encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)

         # sample input
         sample_hidden = encoder.initialize_hidden_state()
         sample_output, sample_hidden = encoder(example_input_batch, sample_hidden)
         print ('Encoder output shape: (batch size, sequence length, units) {}'.format(sample_output.shape))
         print ('Encoder Hidden state shape: (batch size, units) {}'.format(sample_hidden.shape))

         Encoder output shape: (batch size, sequence length, units) (64, 16, 1024)
         Encoder Hidden state shape: (batch size, units) (64, 1024)
```

## Step 12 Defining the Attention Layer

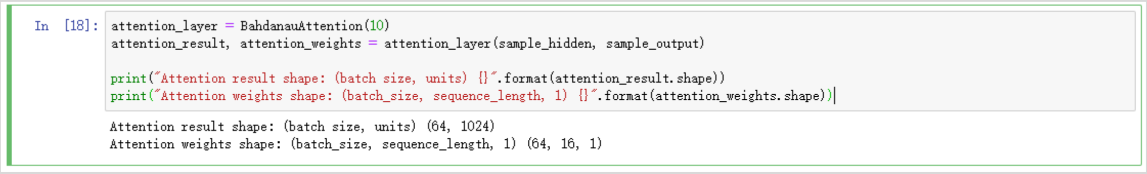
Input:

```
class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, query, values):
        # query hidden state shape == (batch_size, hidden size)
        # query_with_time_axis shape == (batch_size, 1, hidden size)
        # values shape == (batch_size, max_len, hidden size)
        # we are doing this to broadcast addition along the time axis to calculate the score
        query_with_time_axis = tf.expand_dims(query, 1)

        # score shape == (batch_size, max_length, 1)
        # we get 1 at the last axis because we are applying score to self.V
        # the shape of the tensor before applying self.V is (batch_size, max_length, units)
        score = self.V(tf.nn.tanh(
            self.W1(query_with_time_axis) + self.W2(values)))

        # attention_weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

Input:

```
attention_layer = BahdanauAttention(10)
attention_result, attention_weights = attention_layer(sample_hidden, sample_output)

print("Attention result shape: (batch size, units) {}".format(attention_result.shape))
print("Attention weights shape: (batch_size, sequence_length, 1) {}".format(attention_weights.shape))
```

Output:



```
In [18]: attention_layer = BahdanauAttention(10)
         attention_result, attention_weights = attention_layer(sample_hidden, sample_output)

         print("Attention result shape: (batch size, units) {}".format(attention_result.shape))
         print("Attention weights shape: (batch_size, sequence_length, 1) {}".format(attention_weights.shape))

         Attention result shape: (batch size, units) (64, 1024)
         Attention weights shape: (batch_size, sequence_length, 1) (64, 16, 1)
```

### Step 13 Defining a Decoder

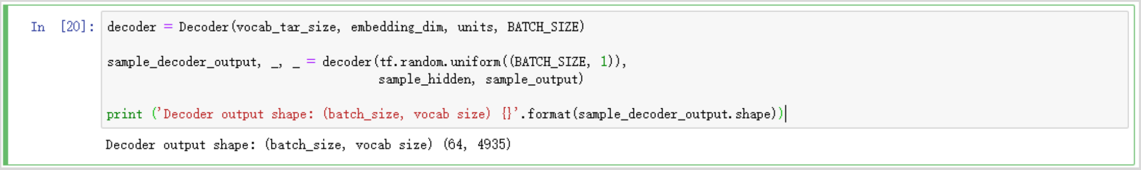
Input:

```
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()
        self.batch_sz = batch_sz
        self.dec_units = dec_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.dec_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
        self.fc = tf.keras.layers.Dense(vocab_size)

        # used for attention
        self.attention = BahdanauAttention(self.dec_units)

    def call(self, x, hidden, enc_output):
        # enc_output shape == (batch_size, max_length, hidden_size)
        context_vector, attention_weights = self.attention(hidden, enc_output)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # output shape == (batch_size * 1, hidden_size)
        output = tf.reshape(output, (-1, output.shape[2]))

        # output shape == (batch_size, vocab)
        x = self.fc(output)

        return x, state, attention_weights
```

Input:

```
decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)

sample_decoder_output, _, _ = decoder(tf.random.uniform((BATCH_SIZE, 1)),
                                      sample_hidden, sample_output)

print ('Decoder output shape: (batch_size, vocab size) {}'.format(sample_decoder_output.shape))
```

Output:



```
In [20]: decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)

         sample_decoder_output, _, _ = decoder(tf.random.uniform((BATCH_SIZE, 1)),
                                               sample_hidden, sample_output)

         print ('Decoder output shape: (batch_size, vocab size) {}'.format(sample_decoder_output.shape))

         Decoder output shape: (batch_size, vocab size) (64, 4935)
```

## Step 14 Define optimizers and losses

Input:

```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

## Step 15 Setting the checkpoint storage path

Input:

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(optimizer=optimizer,
                                 encoder=encoder,
                                 decoder=decoder)
```

## Step 16 Train model

The operations include:

- Pass the input through the encoder which return encoder output and the encoder hidden state.
- The encoder output, encoder hidden state and the decoder input (which is the start token) is passed to the decoder.
- The decoder returns the predictions and the decoder hidden state.
- The decoder hidden state is then passed back into the model and the predictions are used to calculate the loss.
- Use teacher forcing to decide the next input to the decoder.
- **Teacher forcing** is the technique where the target word is passed as the next input to the decoder.
- The final step is to calculate the gradients and apply it to the optimizer and backpropagate.

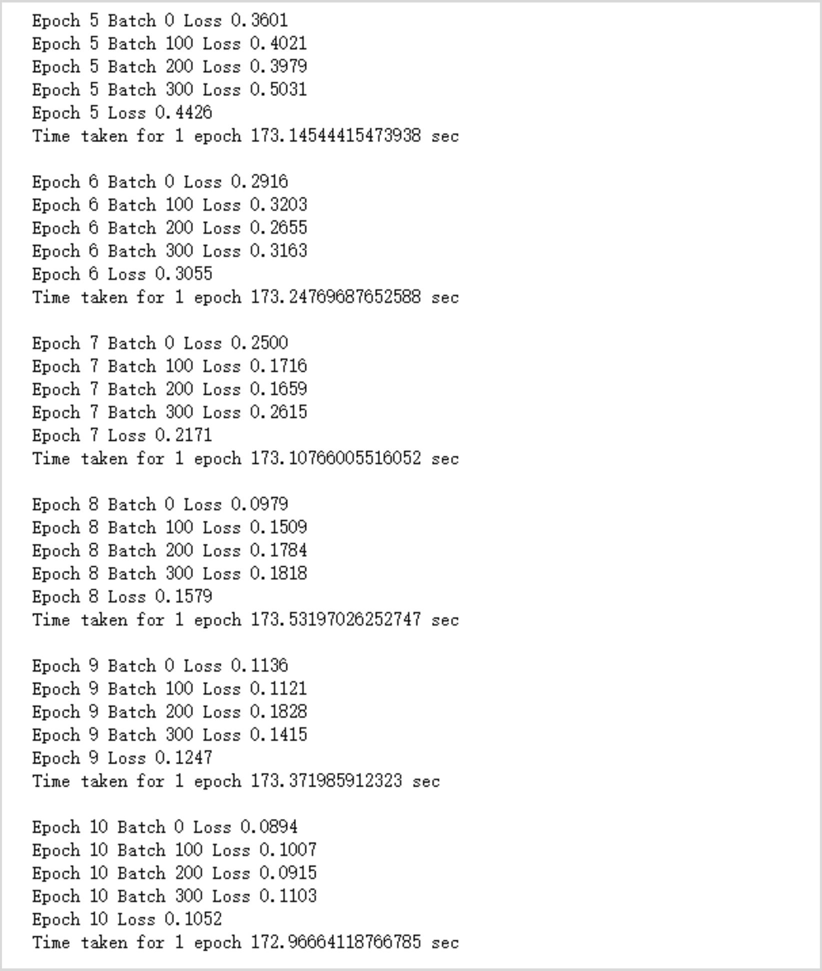
Input:

```
@tf.function
def train_step(inp, targ, enc_hidden):
    loss = 0

    with tf.GradientTape() as tape:
        enc_output, enc_hidden = encoder(inp, enc_hidden)

        dec_hidden = enc_hidden

        dec_input = tf.expand_dims([targ_lang.word_index['<start>']] * BATCH_SIZE, 1)

        # Teacher forcing - feeding the target as the next input
        for t in range(1, targ.shape[1]):
            # passing enc_output to the decoder
            predictions, dec_hidden, _ = decoder(dec_input, dec_hidden, enc_output)

            loss += loss_function(targ[:, t], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(targ[:, t], 1)

    batch_loss = (loss / int(targ.shape[1]))

    variables = encoder.trainable_variables + decoder.trainable_variables

    gradients = tape.gradient(loss, variables)

    optimizer.apply_gradients(zip(gradients, variables))

    return batch_loss
```

Input:

```
EPOCHS = 10

for epoch in range(EPOCHS):
    start = time.time()

    enc_hidden = encoder.initialize_hidden_state()
    total_loss = 0

    for (batch, (inp, targ)) in enumerate(dataset.take(steps_per_epoch)):
        batch_loss = train_step(inp, targ, enc_hidden)
        total_loss += batch_loss

        if batch % 100 == 0:
            print('Epoch {} Batch {} Loss {:.4f}'.format(epoch + 1,
                                                         batch,
                                                         batch_loss.numpy()))
    # saving (checkpoint) the model every 2 epochs
    if (epoch + 1) % 2 == 0:
        checkpoint.save(file_prefix = checkpoint_prefix)

    print('Epoch {} Loss {:.4f}'.format(epoch + 1,
                                        total_loss / steps_per_epoch))
    print('Time taken for 1 epoch {} sec\n'.format(time.time() - start))
```

Output:



```
Epoch 5 Batch 0 Loss 0.3601
Epoch 5 Batch 100 Loss 0.4021
Epoch 5 Batch 200 Loss 0.3979
Epoch 5 Batch 300 Loss 0.5031
Epoch 5 Loss 0.4426
Time taken for 1 epoch 173.14544415473938 sec

Epoch 6 Batch 0 Loss 0.2916
Epoch 6 Batch 100 Loss 0.3203
Epoch 6 Batch 200 Loss 0.2655
Epoch 6 Batch 300 Loss 0.3163
Epoch 6 Loss 0.3055
Time taken for 1 epoch 173.24769687652588 sec

Epoch 7 Batch 0 Loss 0.2500
Epoch 7 Batch 100 Loss 0.1716
Epoch 7 Batch 200 Loss 0.1659
Epoch 7 Batch 300 Loss 0.2615
Epoch 7 Loss 0.2171
Time taken for 1 epoch 173.10766005516052 sec

Epoch 8 Batch 0 Loss 0.0979
Epoch 8 Batch 100 Loss 0.1509
Epoch 8 Batch 200 Loss 0.1784
Epoch 8 Batch 300 Loss 0.1818
Epoch 8 Loss 0.1579
Time taken for 1 epoch 173.53197026252747 sec

Epoch 9 Batch 0 Loss 0.1136
Epoch 9 Batch 100 Loss 0.1121
Epoch 9 Batch 200 Loss 0.1828
Epoch 9 Batch 300 Loss 0.1415
Epoch 9 Loss 0.1247
Time taken for 1 epoch 173.371985912323 sec

Epoch 10 Batch 0 Loss 0.0894
Epoch 10 Batch 100 Loss 0.1007
Epoch 10 Batch 200 Loss 0.0915
Epoch 10 Batch 300 Loss 0.1103
Epoch 10 Loss 0.1052
Time taken for 1 epoch 172.96664118766785 sec
```

### Step 17 Defining test and visualization functions

Input:

```
def evaluate(sentence):
    attention_plot = np.zeros((max_length_targ, max_length_inp))

    sentence = preprocess_sentence(sentence)

    inputs = [inp_lang.word_index[i] for i in sentence.split(' ')]
    inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs],
                                                            maxlen=max_length_inp,
                                                            padding='post')
    inputs = tf.convert_to_tensor(inputs)

    result = ''

    hidden = [tf.zeros((1, units))]
    enc_out, enc_hidden = encoder(inputs, hidden)
    dec_hidden = enc_hidden
    dec_input = tf.expand_dims([targ_lang.word_index['<start>']], 0)

    for t in range(max_length_targ):
        predictions, dec_hidden, attention_weights = decoder(dec_input,
                                                             dec_hidden,
                                                             enc_out)

        # storing the attention weights to plot later on
        attention_weights = tf.reshape(attention_weights, (-1, ))
        attention_plot[t] = attention_weights.numpy()

        predicted_id = tf.argmax(predictions[0]).numpy()

        result += targ_lang.index_word[predicted_id] + ' '

        if targ_lang.index_word[predicted_id] == '<end>':
            return result, sentence, attention_plot

        # the predicted ID is fed back into the model
        dec_input = tf.expand_dims([predicted_id], 0)

    return result, sentence, attention_plot

# function for plotting the attention weights
def plot_attention(attention, sentence, predicted_sentence):
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(1, 1, 1)
    ax.matshow(attention, cmap='viridis')

    fontdict = {'fontsize': 14}

    ax.set_xticklabels([''] + sentence, fontdict=fontdict, rotation=90)
    ax.set_yticklabels([''] + predicted_sentence, fontdict=fontdict)

    ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

    plt.show()

def translate(sentence):
    result, sentence, attention_plot = evaluate(sentence)

    print('Input: %s' % (sentence))
    print('Predicted translation: {}'.format(result))

    attention_plot = attention_plot[:len(result.split(' ')), :len(sentence.split(' '))]
    plot_attention(attention_plot, sentence.split(' '), result.split(' '))
```

### Step 18 Loading a model offline

Input:

```
# restoring the latest checkpoint in checkpoint_dir
```

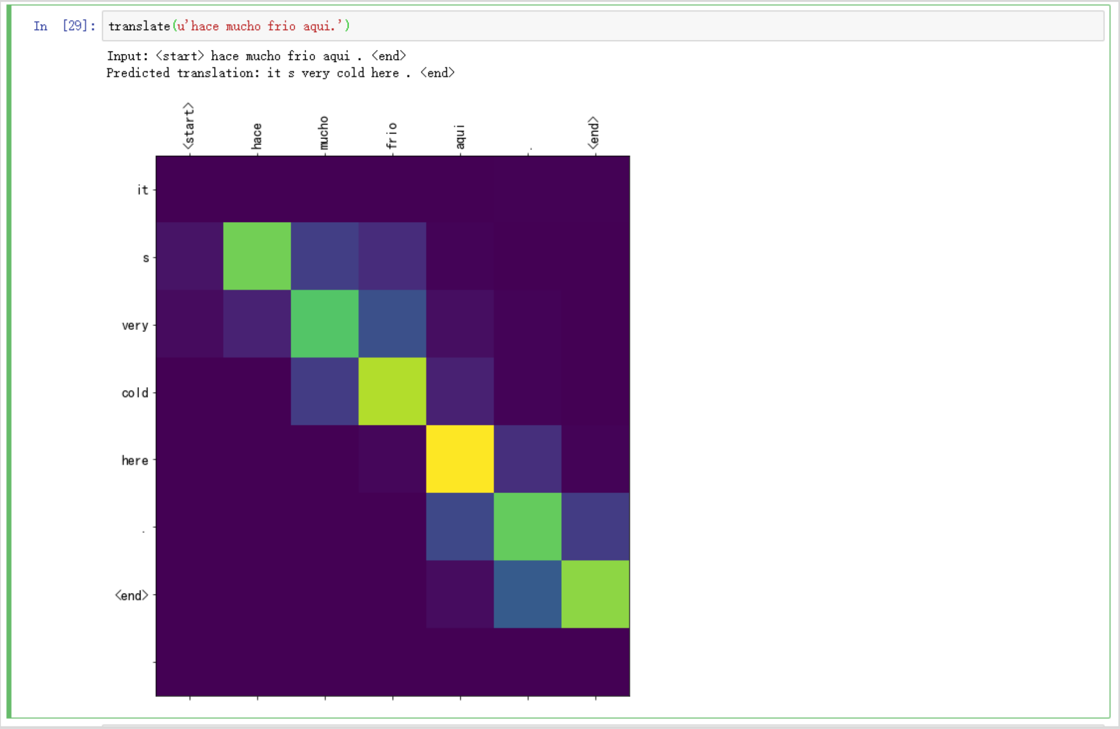
```
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

## Step 19 Single sentence translation test

Input:

```
translate(u'hace mucho frio aqui.')
```

Output:

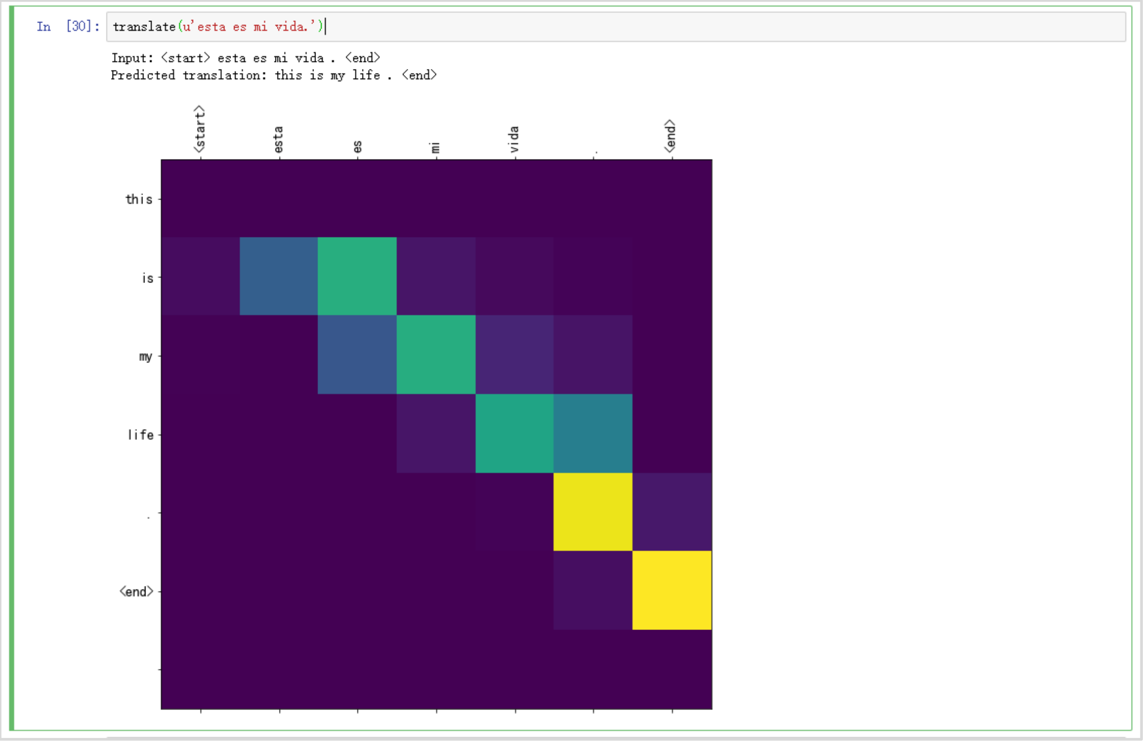


```
In  [29]: translate(u'hace mucho frio aqui.')

          Input: <start> hace mucho frio aqui . <end>
          Predicted translation: it s very cold here . <end>
```

Input:

```
translate(u'esta es mi vida.')
```

Output:



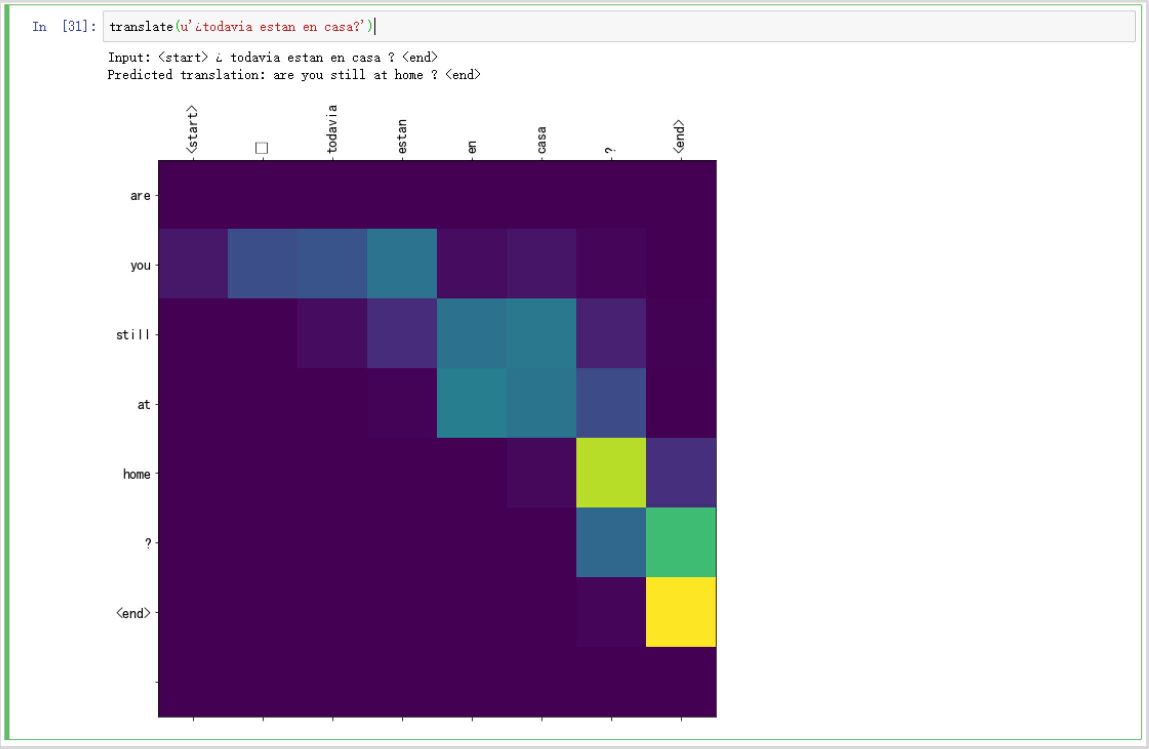
```
In [30]: translate(u'esta es mi vida.')
```

```
Input: <start> esta es mi vida . <end>
Predicted translation: this is my life . <end>
```

Input:

```
translate(u'¿todavia estan en casa?')
```

Output:



```
In [31]: translate(u'¿todavia estan en casa?')
```

```
Input: <start> ¿ todavia estan en casa ? <end>
Predicted translation: are you still at home ? <end>
```

## 3.4 Experiment Summary

This experiment describes how to use tensorflow to build a machine translation model based on the encoder-decoder architecture and the attention mechanism. This experiment helps trainees better understand the encoder-decoder architecture and principles of the attention mechanism, and improves programming practice.

Huawei AI Certification Training

# HCIP-AI-EI Developer

# ModelArts Lab Guide

ISSUE:2.0

HUAWEI TECHNOLOGIES CO., LTD.

**Trademarks and Permissions**

**Notice**

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

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

**Huawei Technologies Co., Ltd.**

Address: Huawei Industrial Base Bantian, Longgang Shenzhen 518129
People's Republic of China

Website: http://e.huawei.com

# Huawei Certificate System

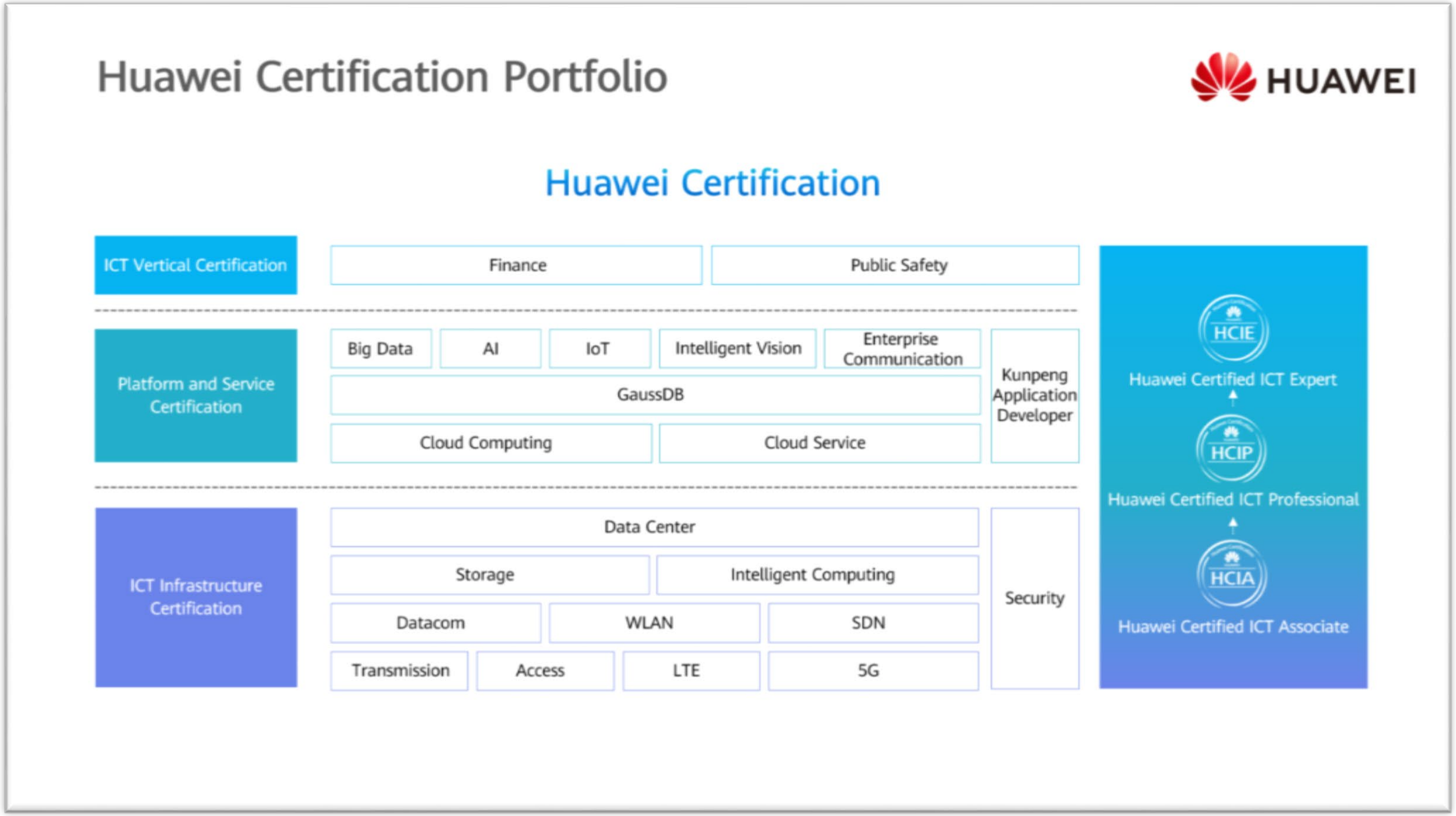
Huawei's certification system is the industry's only one that covers all ICT technical fields. It is developed relying on Huawei's 'platform + ecosystem' strategy and new ICT technical architecture featuring cloud-pipe-device synergy. It provides three types of certifications: ICT Infrastructure Certification, Platform and Service Certification, and ICT Vertical Certification.

To meet ICT professionals' progressive requirements, Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

HCIP-AI-EI Developer V2.0 certification is intended to cultivate professionals who have acquired basic theoretical knowledge about image processing, speech processing, and natural language processing and who are able to conduct development and innovation using Huawei enterprise AI solutions (such as HUAWEI CLOUD EI), general open-source frameworks, and ModelArts, a one-stop development platform for AI developers.

The content of HCIP-AI-EI Developer V2.0 certification includes but is not limited to: neural network basics, image processing theory and applications, speech processing theory and applications, natural language processing theory and applications, ModelArts overview, and image processing, speech processing, natural language processing, and ModelArts platform development experiments. ModelArts is a one-stop development platform for AI developers. With data preprocessing, semi-automatic data labeling, large-scale distributed training, automatic modeling, and on-demand model deployment on devices, edges, and clouds, ModelArts helps AI developers build models quickly and manage the lifecycle of AI development. Compared with V1.0, HCIP-AI-EI Developer V2.0 adds the ModelArts overview and development experiments. In addition, some new EI cloud services are updated.

HCIP-AI-EI Developer V2.0 certification proves that you have systematically understood and mastered neural network basics, image processing theory and applications, speech processing theory and applications, ModelArts overview, natural language processing theory and applications, image processing application development, speech processing application development, natural language processing application development, and ModelArts platform development. With this certification, you will acquire (1) the knowledge and skills for AI pre-sales technical support, AI after-sales technical support, AI product sales, and AI project management; (2) the ability to serve as an image processing developer, speech processing developer, or natural language processing developer.



Huawei Certification Portfolio

HUAWEI

Huawei Certification

ICT Vertical Certification

Finance

Public Safety

Platform and Service Certification

Big Data

AI

IoT

Intelligent Vision

Enterprise Communication

GaussDB

Cloud Computing

Cloud Service

Kunpeng Application Developer

ICT Infrastructure Certification

Data Center

Storage

Intelligent Computing

Datacom

WLAN

SDN

Transmission

Access

LTE

5G

Security

HCIE

Huawei Certified ICT Expert

HCIP

Huawei Certified ICT Professional

HCIA

Huawei Certified ICT Associate

# About This Document

## Overview

This document is intended for trainees who are to take the HCIP-AI certification examination and those who want to learn basic AI knowledge. After completing the experiments in this document, you will be able to understand the AI development lifecycle, and learn how to use ModelArts to develop AI applications, including data uploading, data labeling, deep learning algorithm development, model training, model deployment, and inference. ModelArts is a one-stop AI development platform that provides a wide range of AI development tools. ExeML enables you to quickly build AI applications without coding. Data Management provides data labeling and dataset version management functions. Built-in algorithms can lower the threshold for AI beginners to use the service. Custom deep learning algorithms help you program, train, and deploy AI algorithms.

## Description

This document introduces the following experiments, involving image classification and object detection algorithms based on TensorFlow and MXNet deep learning engines, to help you master practical capabilities of building AI applications.

- Experiment 1: ExeML — Flower Recognition Application
- Experiment 2: ExeML — Yunbao Detection Application
- Experiment 3: ExeML — Bank Deposit Application
- Experiment 4: Data Management — Data Labeling for Flower Recognition
- Experiment 5: Data Management — Data Labeling for Yunbao Detection
- Experiment 6: Data Management — Uploading an MNIST Dataset to OBS
- Experiment 7: Built-in Algorithms — Flower Recognition Application
- Experiment 8: Built-in Algorithms — Yunbao Detection Application
- Experiment 9: Custom Algorithms — Using Native TensorFlow for Handwritten Digit Recognition
- Experiment 10: Custom Algorithms — Using MoXing-TensorFlow for Flower Recognition
- Experiment 11: Custom Algorithms — Using Native MXNet for Handwritten Digit Recognition
- Experiment 12: Custom Algorithms — Using MoXing-MXNet for Flower Recognition

# Background Knowledge Required

This course is for Huawei's development certification. To better understand this course, familiarize/equip yourself with the following:

- Basic language editing capabilities
- Data structure basics
- Python programming basics
- Basic deep learning concepts
- Basic TensorFlow and MXNet concepts

# Experiment Environment Overview

ModelArts provides a cloud-based development environment. You do not need to install one.

# Experiment Data Overview

Download the datasets and source code used in this document from https://huawei-ai-certification.obs.cn-north-4.myhuaweicloud.com/ENG/HCIP-ModelArts%20V2.1.rar

# Contents

# 1 ExeML

## 1.1 About This Lab

ExeML, a service provided by ModelArts, is the process of automating model design, parameter tuning and training, and model compression and deployment with the labeled data. The process is free of coding and does not require your experience in model development, enabling you to start from scratch. This lab guides you through image classification, object detection, and predictive analytics scenarios.

Image classification is based on image content labeling. An image classification model can predict a label corresponding to an image, and is applicable to scenarios in which image classes are obvious. In addition to predicting class labels in images, an object detection model can also predict objects' location information, and is suitable for complex image detection scenarios. A predictive analytics model is used to classify structured data or predict values, which can be used in structured data predictive analysis scenarios.

## 1.2 Objectives

This lab uses three specific examples to help you quickly create image classification, object detection, and predictive analytics models. The flower recognition experiment recognizes flower classes in images. The Yunbao detection experiment identifies Yunbaos' locations and actual classes in images. The bank deposit prediction experiment classifies or predicts values of structured data. After doing these three experiments, you can quickly understand the scenarios and usage of image classification, object detection, and predictive analytics models.

## 1.3 Experiment Environment Overview

If you are a first-time ModelArts user, you need to add an access key to authorize ModelArts jobs to access Object Storage Service (OBS) on HUAWEI CLOUD. You cannot create any jobs without an access key. The procedure is as follows:

- Generating an access key: On the management console, move your cursor over your username, and choose Basic Information > Manage > My Credentials > Access Keys to create an access key. After the access key is created, the AK/SK file will be downloaded to your local computer.

- Configuring global settings for ModelArts: Go to the Settings page of ModelArts, and enter the AK and SK information recorded in the downloaded AK/SK file to authorize ModelArts modules to access OBS.

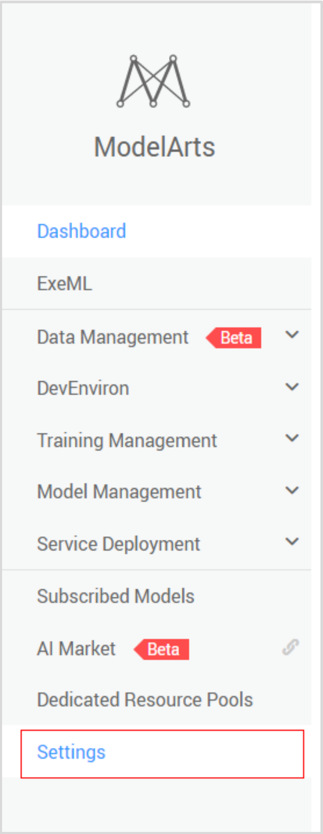


**Figure 1-1 ModelArts management console**

# 1.4 Procedure

## 1.4.1 Flower Recognition Application

The **ExeML** page consists of two parts. The upper part lists the supported ExeML project types. You can click **Create Project** to create an ExeML project. The created ExeML projects are listed in the lower part of the page. You can filter the projects by type or search for a project by entering its name in the search box and clicking 🔍.

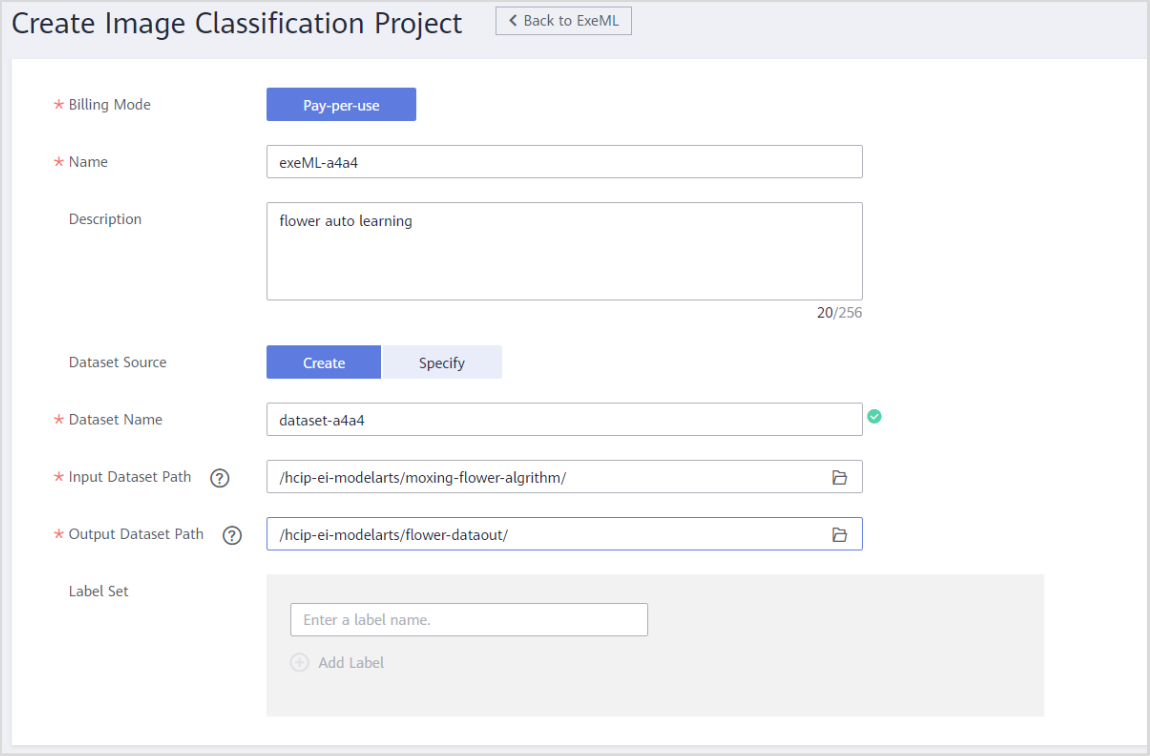
The procedure for using ExeML is as follows:

- Creating a project: To use ModelArts ExeML, create an ExeML project first.
- Labeling data: Upload images and label them by class.
- Training a model: After data labeling is completed, you can start model training.
- Deploying a service and performing prediction: Deploy the trained model as a service and perform online prediction.

## 1.4.2 Creating a Project

Step 1 Create a project.

On the **ExeML** page, click **Create Project** in **Image Classification**. The **Create Image Classification Project** page is displayed. See Figure 1-2.



Create Image Classification Project

* Billing Mode: Pay-per-use

* Name: exeML-a4a4

Description: flower auto learning

20/256

Dataset Source: Create | Specify

* Dataset Name: dataset-a4a4

* Input Dataset Path: /hcip-ei-modelarts/moxing-flower-algrithm/

* Output Dataset Path: /hcip-ei-modelarts/flower-dataout/

Label Set: Enter a label name.

**Figure 1-2 Creating a project**

Parameters:

Billing Mode: Pay-per-use by default

**Name**: The value can be modified as required.

**Input Dataset Path**: Select an OBS path for storing the dataset to be trained. Create an empty folder on OBS first (Click the bucket name to enter the bucket. Then, click **Create Folder**, enter a folder name, and click **OK**). Select the newly created OBS folder as the training data path. Alternatively, you can import required data to OBS in advance. In this example, the data is uploaded to the **/modelarts-demo/auto-learning/image-class** folder. For details about how to upload data, see https://support.huaweicloud.com/en-us/modelarts_faq/modelarts_05_0013.html. To obtain the source data, visit **modelarts-datasets-and-source-code/ExeML/flower-recognition-application/training-dataset**.

**Description**: The value can be modified as required.

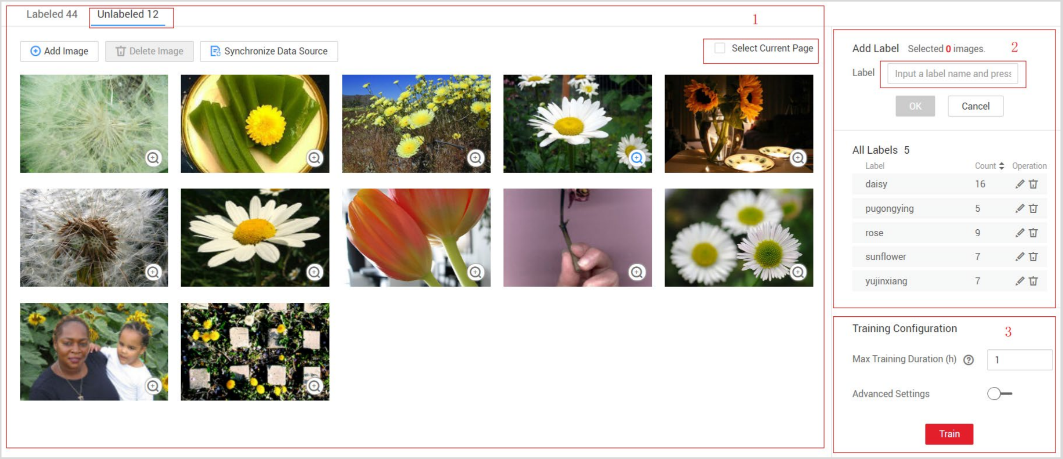
Step 2 Confirm the project creation.

Click **Create Project**. The ExeML project is created.

### 1.4.2.2 Labeling Data

Step 1 Upload images.

After an ExeML project is created, the **Label Data** page is automatically displayed. Click **Add Image** to add images in batches. The dataset path is **modelarts-datasets-and-source-code/ExeML/flower-recognition-application/training-dataset**. If the images have been uploaded to OBS, click **Synchronize Data Source** to synchronize the images to ModelArts. See Figure 1-3.



Add Label Selected 0 images.

Label: Input a label name and press

All Labels 5

| Label | Count | Operation |
|---|---|---|
| daisy | 16 | ✎ 🗑 |
| pugongying | 5 | ✎ 🗑 |
| rose | 9 | ✎ 🗑 |
| sunflower | 7 | ✎ 🗑 |
| yujinxiang | 7 | ✎ 🗑 |

Training Configuration

Max Training Duration (h): 1

Advanced Settings: ○—

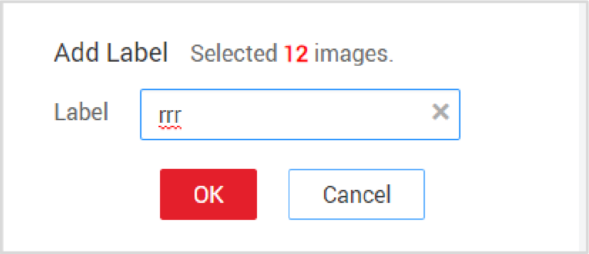
**Figure 1-3 Data labeling page of an image classification project**

**NOTE**

- The images to be trained must be classified into at least two classes, and each class must contain at least five images. That is, at least two labels are available and the number of images for each label is not fewer than five.
- You can add multiple labels to an image.

Step 2 Label the images.

In area 1, click **Unlabeled**, and select one or more images to be labeled in sequence, or select **Select Current Page** in the upper right corner to select all images on the current page. In area 2, input a label or select an existing label and press **Enter** to add the label to the images. Then, click **OK**. The selected images are labeled. See Figure 1-4.



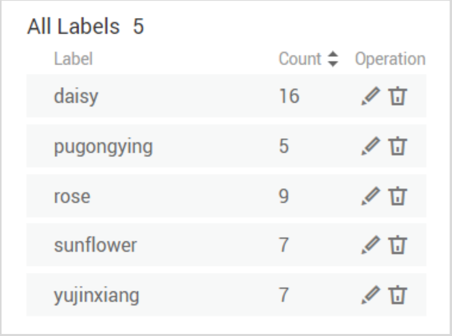
Add Label Selected 12 images.

Label: rrr

**Figure 1-4 Image labeling for image classification**

Step 3 Delete or modify a label in one image.

Click **Labeled** in area 1, and then click an image. To modify a label, click ✎ on the right of the label in area 2, enter a new label on the displayed dialog box, and click ✓. To delete a label, click 🗑 on the right of the label in area 2. See Figure 1-5.



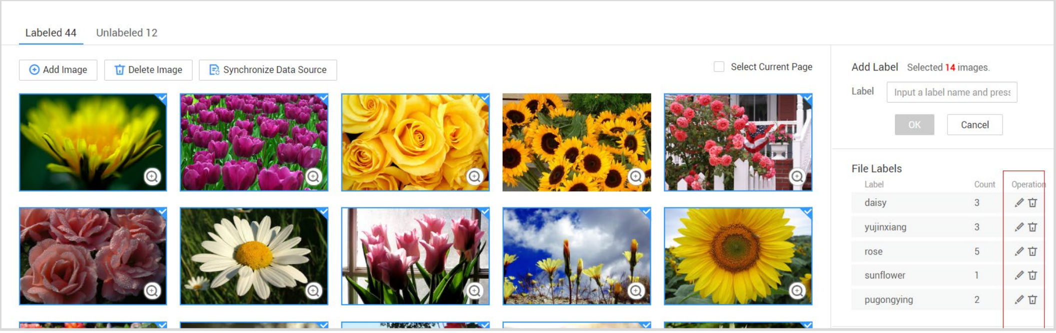
All Labels 5

| Label | Count ⇕ | Operation |
|---|---|---|
| daisy | 16 | ✎ 🗑 |
| pugongying | 5 | ✎ 🗑 |
| rose | 9 | ✎ 🗑 |
| sunflower | 7 | ✎ 🗑 |
| yujinxiang | 7 | ✎ 🗑 |

**Figure 1-5 Deleting/Modifying a label in one image**

Step 4 Delete or modify a label in multiple images.

In area 2, click the label to be modified or deleted, and click ✎ on the right of the label to rename it, or click 🗑 to delete it from multiple images. In the dialog box that is displayed, select **Delete label** or **Delete label and images that only contain this label**. See Figure 1-6.



Add Label Selected 14 images.

Label: Input a label name and press

File Labels

| Label | Count | Operation |
|---|---|---|
| daisy | 3 | ✎ 🗑 |
| yujinxiang | 3 | ✎ 🗑 |
| rose | 5 | ✎ 🗑 |
| sunflower | 1 | ✎ 🗑 |
| pugongying | 2 | ✎ 🗑 |

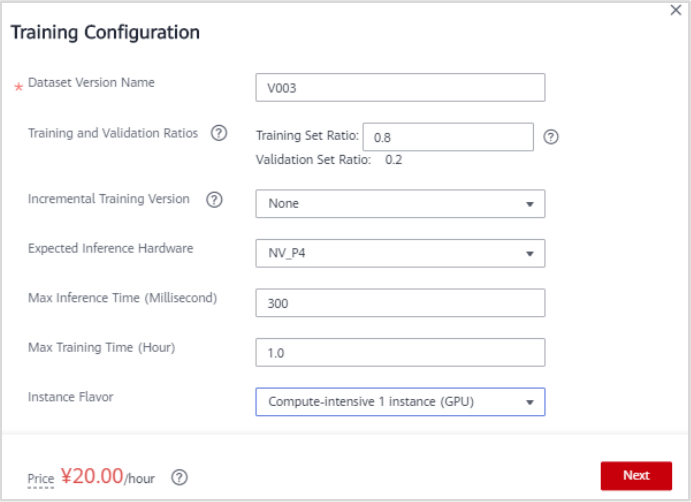
**Figure 1-6 Deleting/Modifying a label in multiple images**

### 1.4.2.3 Training a Model

After labeling the images, you can train an image classification model. Set the training parameters first and then start automatic training of the model. Images to be trained must be classified into at least two classes, and each class must contain at least five images. Therefore, before training, ensure that the labeled images meet the requirements. Otherwise, the **Train** button is unavailable.

Step 1 Set related parameters.

You can retain the default values for the parameters, or modify **Max Training Duration (h)** and enable **Advanced Settings** to set the inference duration. Figure 1-7 shows the training settings.



Training Configuration

| | |
|---|---|
| * Dataset Version Name | V003 |
| Training and Validation Ratios | Training Set Ratio: 0.8<br>Validation Set Ratio: 0.2 |
| Incremental Training Version | None |
| Expected Inference Hardware | NV_P4 |
| Max Inference Time (Millisecond) | 300 |
| Max Training Time (Hour) | 1.0 |
| Instance Flavor | Compute-intensive 1 instance (GPU) |

Price ¥20.00/hour

**Figure 1-7 Training settings**

Parameters:

**Max Training Duration (h)**: If the training process is not completed within the maximum training duration, it is forcibly stopped. You are advised to enter a larger value to prevent forcible stop during training.

**Max Inference Duration (ms)**: The time required for inferring a single image is proportional to the complexity of the model. Generally, the shorter the inference time, the simpler the selected model and the faster the training speed. However, the precision may be affected.

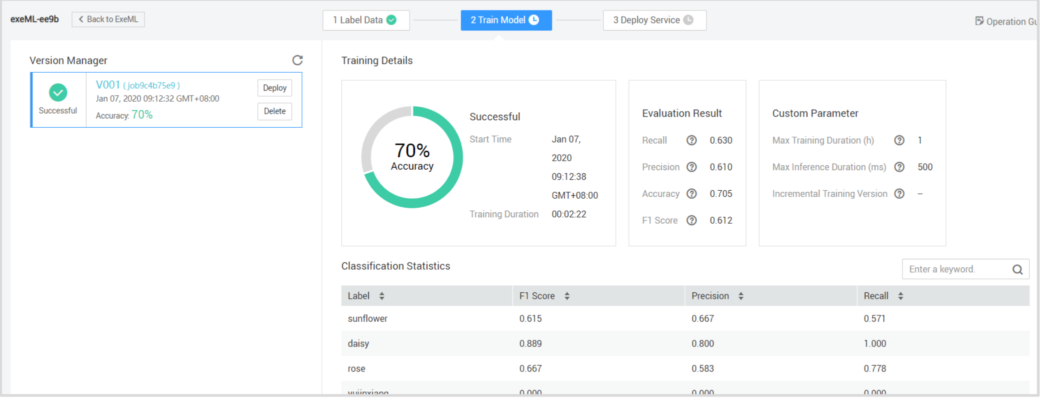
Step 2 Train a model.

After setting the parameters, click **Train**. After training is completed, you can view the training result on the **Train Model** tab page.

### 1.4.2.4 Deploying a Service and Performing Prediction

Step 1 Deploy the model as a service.

After the model training is completed, you can deploy a version with the ideal precision and in the **Successful** status as a service. To do so, click **Deploy** in the **Version Manager** pane of the **Train Model** tab page. See Figure 1-8. After the deployment is successful, you can choose **Service Deployment** > **Real-Time Services** to view the deployed service.



exeML-ee9b < Back to ExeML

1 Label Data ✓ — 2 Train Model — 3 Deploy Service

Operation Gu

Version Manager

Successful V001 ( job9c4b75e9 ) Jan 07, 2020 09:12:32 GMT+08:00 Accuracy: 70% Deploy Delete

Training Details

70% Accuracy

Successful

Start Time Jan 07, 2020 09:12:38 GMT+08:00

Training Duration 00:02:22

Evaluation Result

| | |
|---|---|
| Recall | 0.630 |
| Precision | 0.610 |
| Accuracy | 0.705 |
| F1 Score | 0.612 |

Custom Parameter

| | |
|---|---|
| Max Training Duration (h) | 1 |
| Max Inference Duration (ms) | 500 |
| Incremental Training Version | -- |

Classification Statistics

Enter a keyword.

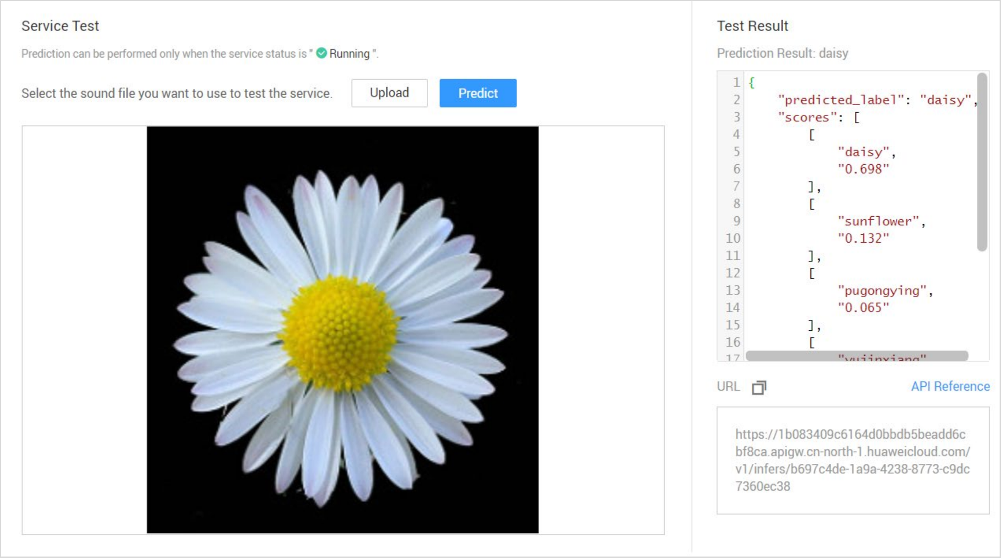
| Label | F1 Score | Precision | Recall |
|---|---|---|---|
| sunflower | 0.615 | 0.667 | 0.571 |
| daisy | 0.889 | 0.800 | 1.000 |
| rose | 0.667 | 0.583 | 0.778 |
| yujinxiang | 0.000 | 0.000 | 0.000 |

Figure 1-8 Deploying the model as a service

Step 2 Test the service.

After the model is deployed as a service, you can upload an image to test the service. The path of the test data is **modelarts-datasets-and-source-code/ExeML/flower-recognition-application/test-data/daisy.jpg**.

On the **Deploy Service** tab page, click the **Upload** button to select the test image. After the image is uploaded successfully, click **Predict**. The prediction result is displayed in the right pane. See Figure 1-9. Five classes of labels are added during data labeling: tulip, daisy, sunflower, rose, and dandelion. The test image contains a daisy. In the prediction result, "daisy" gets the highest score, that is, the classification result is "daisy".



Service Test

Prediction can be performed only when the service status is " ✓ Running ".

Select the sound file you want to use to test the service. Upload Predict

Test Result

Prediction Result: daisy

```
{
    "predicted_label": "daisy",
    "scores": [
        [
            "daisy",
            "0.698"
        ],
        [
            "sunflower",
            "0.132"
        ],
        [
            "pugongying",
            "0.065"
        ],
        [
            "yujinxiang"
```

URL API Reference

https://1b083409c6164d0bbdb5beadd6cbf8ca.apigw.cn-north-1.huaweicloud.com/v1/infers/b697c4de-1a9a-4238-8773-c9dc7360ec38

Figure 1-9 Service testing

## 1.4.3 Yunbao Detection Application

The **ExeML** page consists of two parts. The upper part lists the supported ExeML project types. You can click **Create Project** to create an ExeML project. The created ExeML projects are listed in the lower part of the page. You can filter the projects by type or search for a project by entering its name in the search box and clicking 🔍.

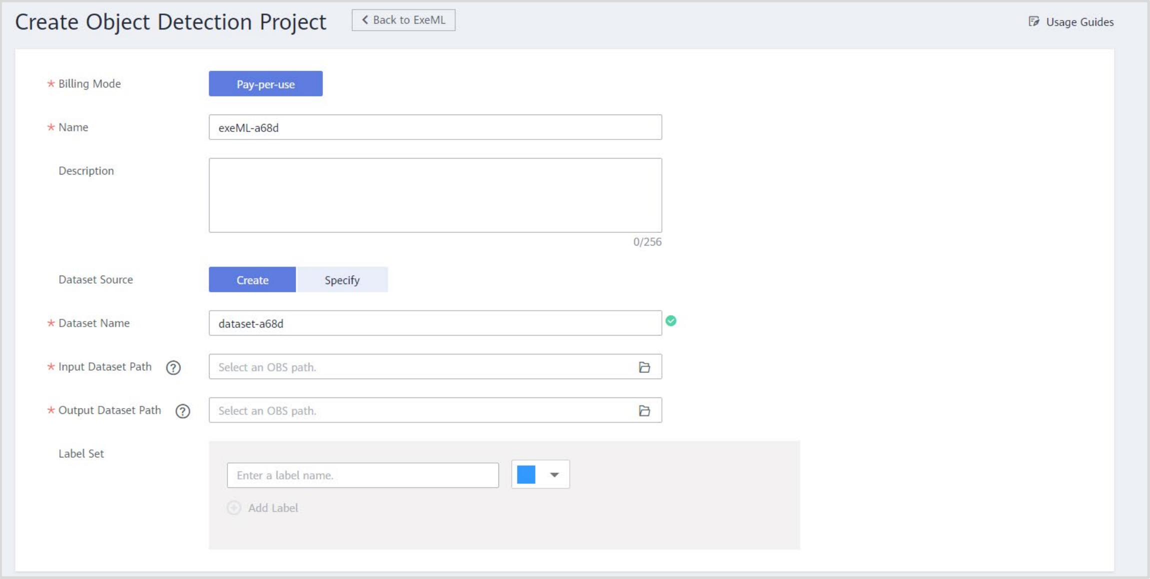
The procedure for using ExeML is as follows:

- Creating a project: To use ModelArts ExeML, create an ExeML project first.
- Labeling data: Upload images and label them by class.
- Training a model: After data labeling is completed, you can start model training.
- Deploying a service and performing prediction: Deploy the trained model as a service and perform online prediction.

### 1.4.3.1 Creating a Project

Step 1 Create a project.

On the **ExeML** page, click **Create Project** in **Object Detection**. The **Create Object Detection Project** page is displayed. See Figure 1-10.



Create Object Detection Project

* Billing Mode: Pay-per-use

* Name: exeML-a68d

Description: 0/256

Dataset Source: Create | Specify

* Dataset Name: dataset-a68d

* Input Dataset Path: Select an OBS path.

* Output Dataset Path: Select an OBS path.

Label Set: Enter a label name.

**Figure 1-10 Creating a project.**

Parameters:

Billing Mode: Pay-per-use by default

**Name**: The value can be modified as required.

**Training Data**: Create an empty folder on OBS and specify the OBS folder path as the value of this parameter. In this example, **/modelarts-demo/auto-learning/object-detection** is used. Alternatively, you can directly import data to OBS in advance. For details, see **2.3.3** "Uploading an MNIST Dataset to OBS."

**Description**: The value can be modified as required.

Step 2 Confirm the project creation.

Click **Create Project**. The ExeML project is created.

### 1.4.3.2 Labeling Data

Step 1 Upload images.

After an ExeML project is created, the **Label Data** page is automatically displayed. Click **Add Image** to add images in batches. Note that the total size of the images uploaded in one attempt cannot exceed 8 MB. The dataset path is **modelarts-datasets-and-source-code/ExeML/yunbao-detection-application/training-dataset**. The dataset contains images of Yunbao, the mascot of HUAWEI CLOUD. If the images have been uploaded to OBS, click **Synchronize Data Source** to synchronize the images to ModelArts. See Figure 1-11.

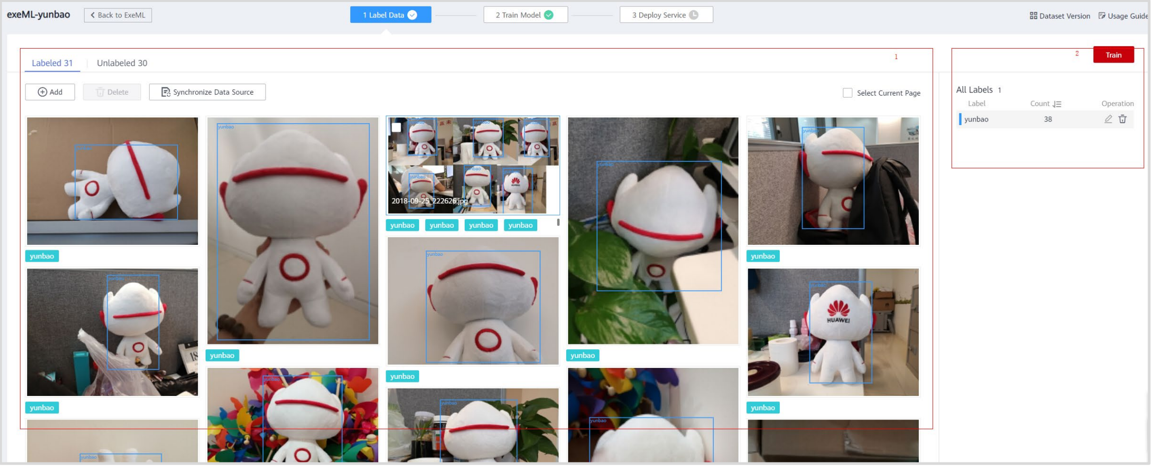


**Figure 1-11 Data labeling page of an object detection project**

NOTE

- Each class of images to be trained must contain at least five images. That is, the number of images for each label is not fewer than five.
- You can add multiple labels to an image.

Step 2 Label the images.

Enter the **Unlabeled** tab page and click an image to access its labeling page. See Figure 1-12. On the labeling page, draw a labeling box to frame out the target object. Ensure that the box does not contain too much background information. Then, select a label. If no label is available, input one and press **Enter**.

In this example, use the mouse to draw a box to frame the Yunbao and input **yunbao** as the label name. See Figure 1-13.

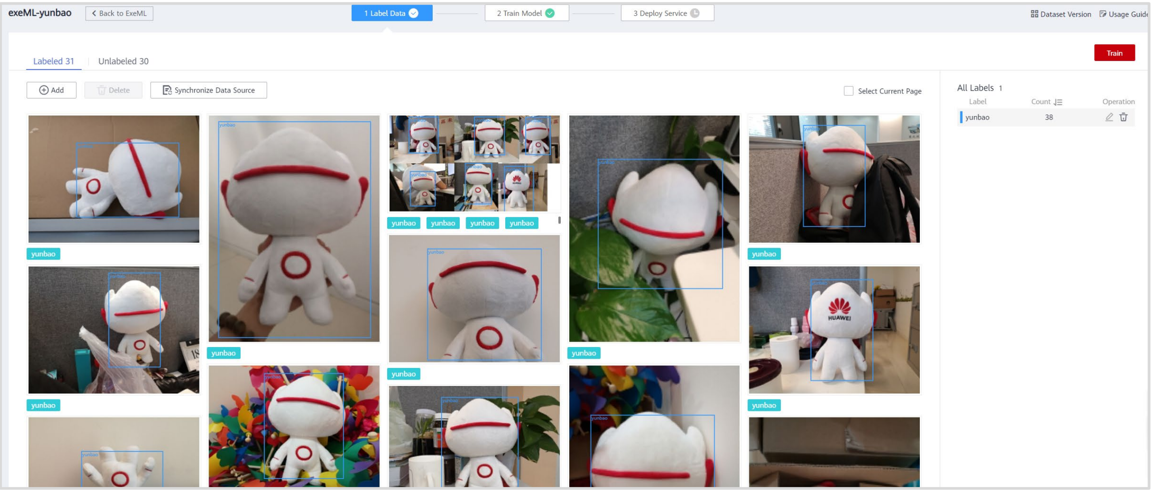
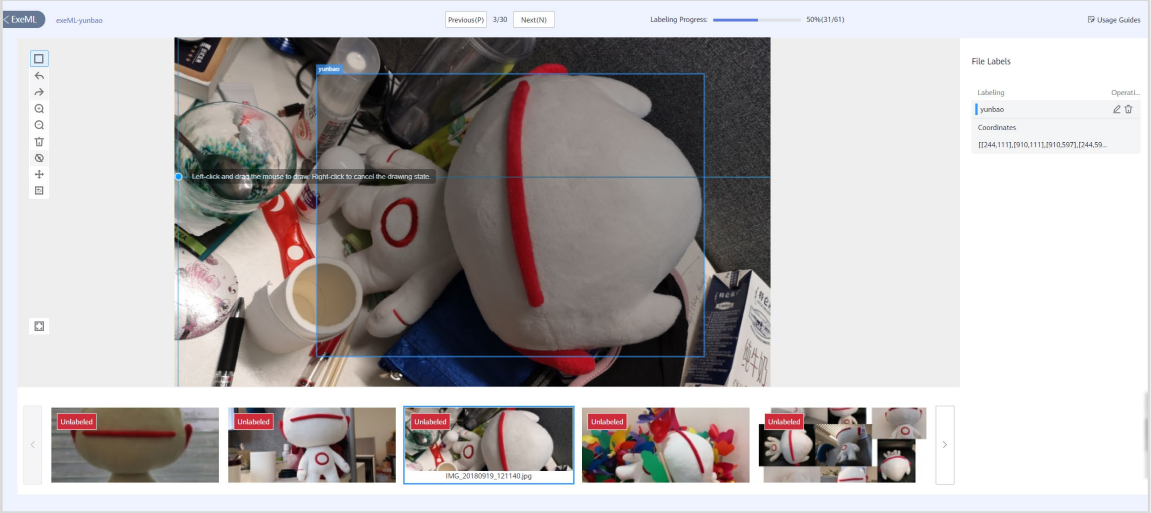


Figure 1-12 Image labeling for object detection



Left-click and drag the mouse to draw. Right-click to cancel the drawing state.

[[244,111],[910,111],[910,597],[244,59...

Figure 1-13 Image labeling page

Step 3 Delete or modify a label in one image.

Click the **Labeled** tab and click the target image to enter its labeling page. Then, you can delete or modify a label through either of the following methods:

- Method 1: Move the cursor to the labeling box, right-click, and choose Modify from the shortcut menu to modify the label or choose Delete to delete the label.

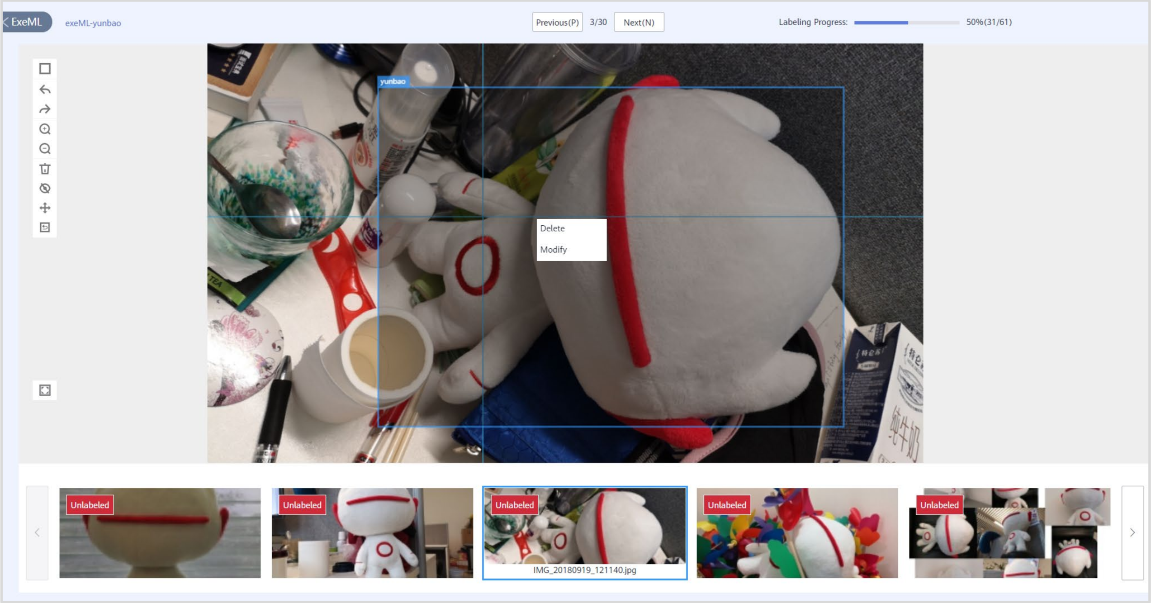


**Figure 1-14 Deleting/Modifying a label in one image**

- Method 2: Click the ✎ or 🗑 button on the right of the image to modify or delete its label.

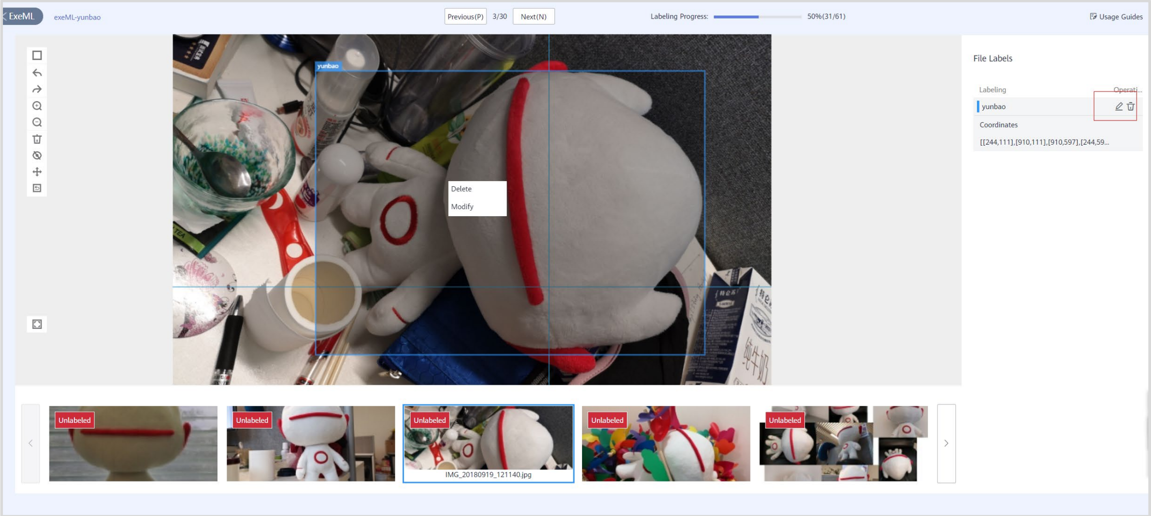


**Figure 1-15 Deleting a label and adding a new label in one image**

Step 4 Delete or modify a label in multiple images.

In area 2 of the **Labeled** tab page, click ✎ on the right of the target label to rename it, or click 🗑 to delete it from multiple images. In the dialog box that is displayed, select **Delete label** or **Delete label and images that only contain this label**. See Figure 1-16.

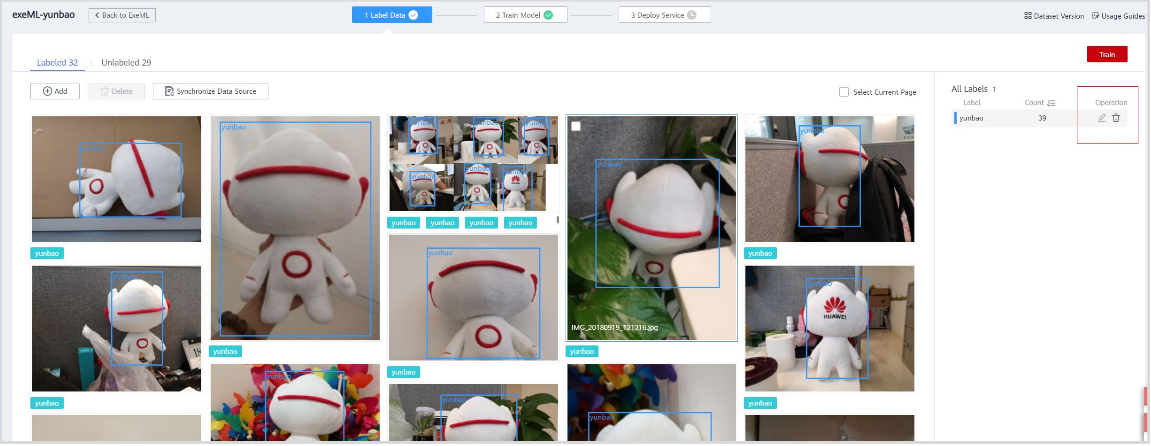


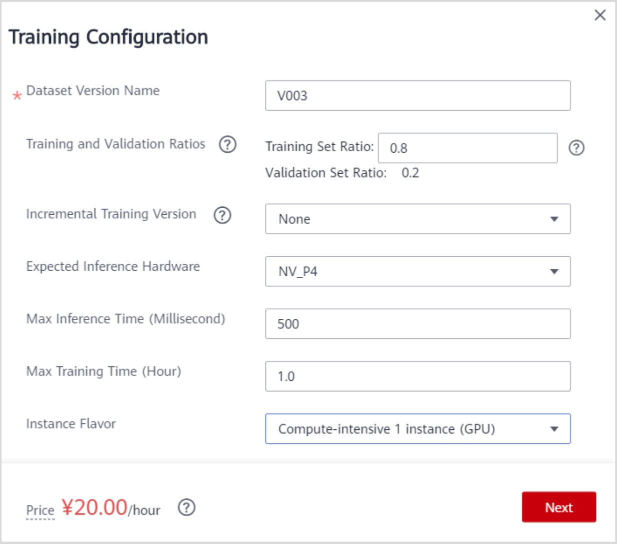
Figure 1-16 Deleting/Modifying a label in multiple images

## 1.4.3.3 Training a Model

After labeling the images, you can train an object detection model. Set the training parameters first and then start automatic training of the model. Each class of images to be trained must contain at least five images. Therefore, before training, ensure that the labeled images meet the requirements. Otherwise, the **Train** button is unavailable.

Step 1 Set the parameters.

You can retain the default values for the parameters, or modify **Max Training Duration (h)** and enable **Advanced Settings** to set the inference duration. Figure 1-17 shows the training settings.


Training Configuration

* Dataset Version Name: V003

Training and Validation Ratios: Training Set Ratio: 0.8 Validation Set Ratio: 0.2

Incremental Training Version: None

Expected Inference Hardware: NV_P4

Max Inference Time (Millisecond): 500

Max Training Time (Hour): 1.0

Instance Flavor: Compute-intensive 1 instance (GPU)

Price ¥20.00/hour

Figure 1-17 Training settings

Parameters:

**Max Training Duration (h)**: If the training process is not completed within the maximum training duration, it is forcibly stopped. You are advised to enter a larger value to prevent forcible stop during training.

**Max Inference Duration (ms)**: The time required for inferring a single image is proportional to the complexity of the model. Generally, the shorter the inference time, the simpler the selected model and the faster the training speed. However, the precision may be affected.

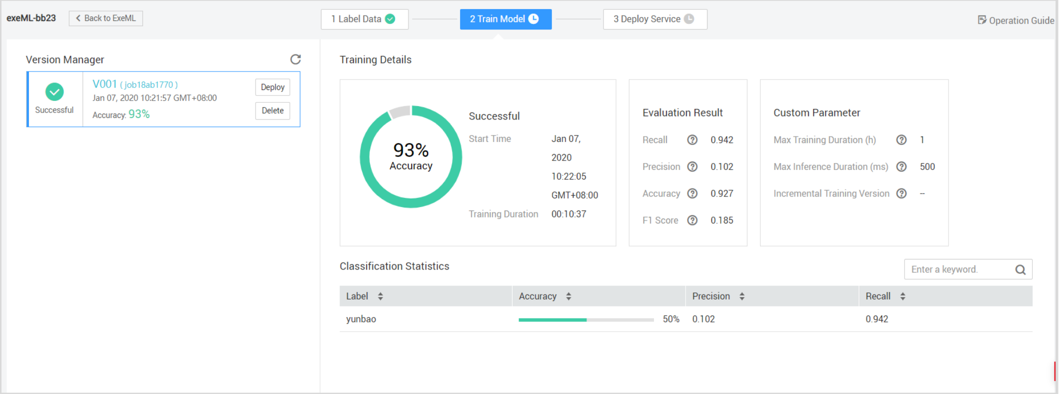
Step 2 Train a model.

After setting the parameters, click **Train**. After training is completed, you can view the training result on the **Train Model** tab page.

### 1.4.3.4 Deploying a Service and Performing Prediction

Step 1 Deploying the model as a service

After the model training is completed, you can deploy a version with the ideal precision and in the **Successful** status as a service. To do so, click **Deploy** in the **Version Manager** pane of the **Train Model** tab page. See Figure 1-18. After the deployment is successful, you can choose **Service Deployment** > **Real-Time Services** to view the deployed service.



V001 ( job18ab1770 )
Jan 07, 2020 10:21:57 GMT+08:00
Accuracy: 93%

Training Details

93% Accuracy

Successful
Start Time Jan 07, 2020 10:22:05 GMT+08:00
Training Duration 00:10:37

Evaluation Result
Recall 0.942
Precision 0.102
Accuracy 0.927
F1 Score 0.185

Custom Parameter
Max Training Duration (h) 1
Max Inference Duration (ms) 500
Incremental Training Version --

Classification Statistics

Enter a keyword.

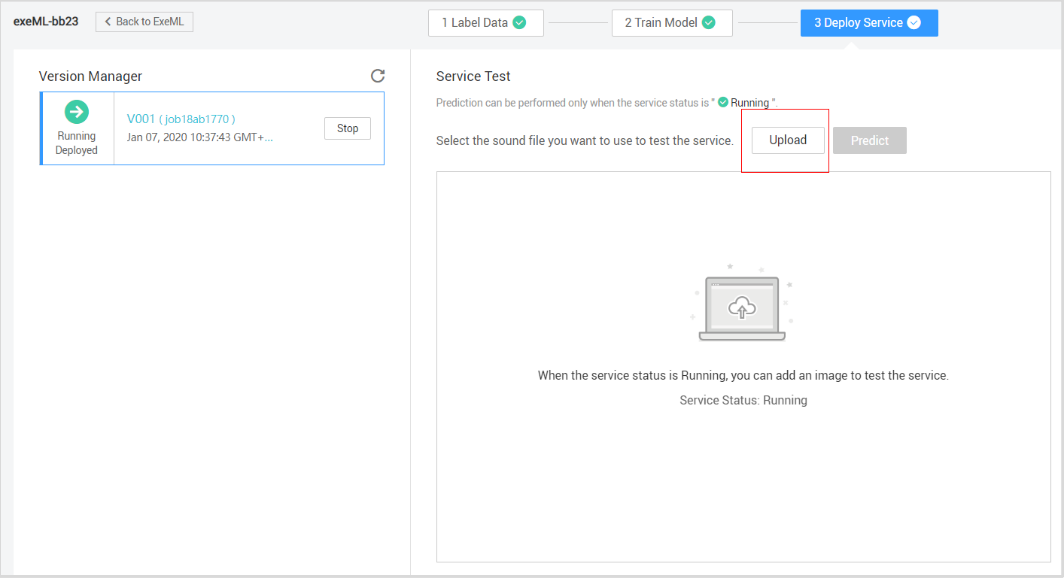
| Label | Accuracy | Precision | Recall |
|---|---|---|---|
| yunbao | 50% | 0.102 | 0.942 |

**Figure 1-18 Deploying the model as a service**

Step 2 Test the service.

After the model is deployed, you can upload an image to test the service. The path of the test data is modelarts-datasets-and-source-code/ExeML/yunbao-detection-application/test-data.

On the Deploy Service tab page, click the Upload button to select the test image. After the image is uploaded successfully, click Predict. The prediction result is displayed in the right pane. See the following figures. In the prediction result, Yunbaos are framed out with boxes and labeled with yunbao, and the related probabilities and coordinate values are displayed in the right pane.



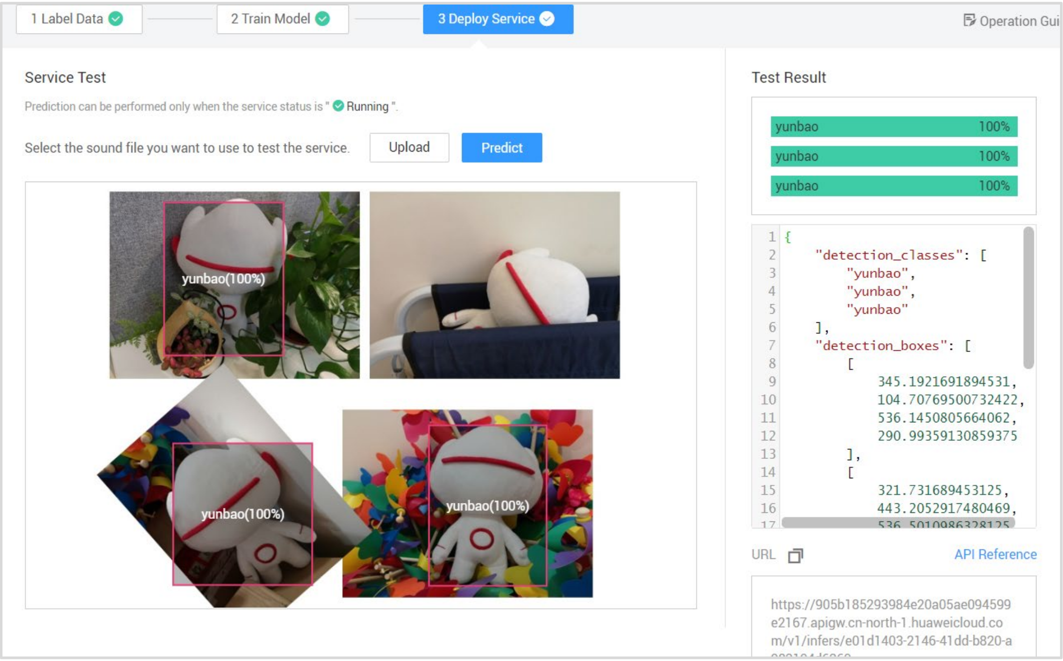
Prediction can be performed only when the service status is " Running ".

Select the sound file you want to use to test the service.

When the service status is Running, you can add an image to test the service.

Service Status: Running

**Figure 1-19 Uploading a test image**



Prediction can be performed only when the service status is " Running ".

Select the sound file you want to use to test the service.

| | |
|---|---|
| yunbao | 100% |
| yunbao | 100% |
| yunbao | 100% |

```
{
    "detection_classes": [
        "yunbao",
        "yunbao",
        "yunbao"
    ],
    "detection_boxes": [
        [
            345.1921691894531,
            104.70769500732422,
            536.1450805664062,
            290.99359130859375
        ],
        [
            321.731689453125,
            443.2052917480469,
            536.5010986328125
```

https://905b185293984e20a05ae094599e2167.apigw.cn-north-1.huaweicloud.com/v1/infers/e01d1403-2146-41dd-b820-a

**Figure 1-20 Service testing**

## 1.4.4 Bank Deposit Prediction Application

This experiment describes how to use ModelArts to predict the bank deposit.

Banks often predict whether customers would be interested in a time deposit based on their characteristics, including the age, work type, marital status, education background, housing loan, and personal loan.

Now, you can use the ExeML function of HUAWEI CLOUD ModelArts to easily predict whether a customer would be interested in the time deposit. The procedure consists of three parts:

- Preparing data: Download a dataset and upload it to OBS.

- Training a model: Use ModelArts to create a project for model training.
- Deploying a service and performing prediction: Deploy the trained model as a service and test the prediction function.

### 1.4.4.1 Preparing Data

To upload the training dataset to an OBS bucket, perform the following steps:

Step 1 Find the train.csv file (training dataset) in the modelarts-datasets-and-source-code/data-management/bank-deposit-prediction-application/dataset directory.

Step 2 Browse and understand the training dataset.

**Table 1-1 Parameters and meanings**

| Parameter | Meaning | Type | Description |
|---|---|---|---|
| attr_1 | Age | Int | Age of the customer |
| attr_2 | Occupation | String | Occupation of the customer |
| attr_3 | Marital status | String | Marital status of the customer |
| attr_4 | Education status | String | Education status of the customer |
| attr_5 | Real estate | String | Real estate of the customer |
| attr_6 | Loan | String | Loan of the customer |
| attr_7 | Deposit | String | Deposit of the customer |

**Table 1-2 Sample data of the dataset**

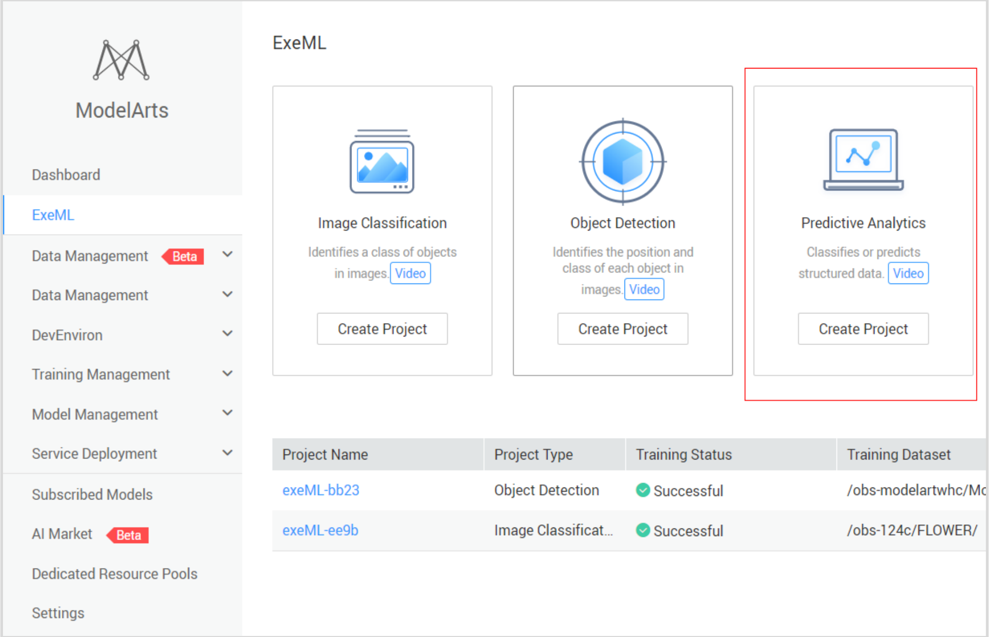
| attr_1 | attr_2 | attr_3 | attr_4 | attr_5 | attr_6 | attr_7 |
|---|---|---|---|---|---|---|
| 58 | management | married | tertiary | yes | no | no |
| 44 | technician | single | secondary | yes | no | no |
| 33 | entrepreneur | married | secondary | yes | yes | no |
| 47 | blue-collar | married | unknown | yes | no | no |
| 33 | unknown | single | unknown | no | no | no |
| 35 | management | married | tertiary | yes | no | no |

Step 3 Upload the training dataset file from your local computer to the OBS bucket. For details about how to upload a file to OBS, see https://support.huaweicloud.com/qs-obs/obs_qs_0001.html.

### 1.4.4.2 Training a Model

To create a project for model training using ModelArts, perform the following steps:

Step 1 Enter the ModelArts management console, and choose **ExeML** > **Predictive Analytics** > **Create Project** to create a predictive analytics project. When creating the project, select the training dataset uploaded to OBS in previous steps.



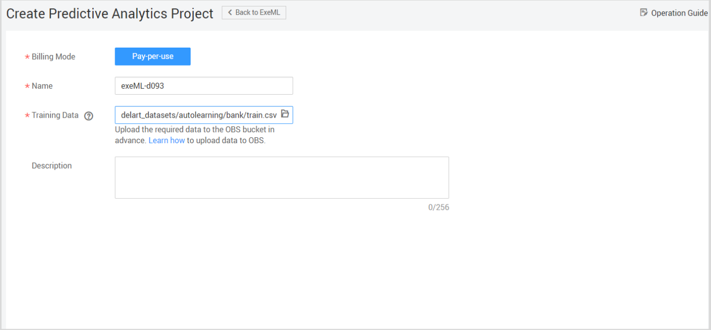
Identifies a class of objects in images.

Identifies the position and class of each object in images.

Classifies or predicts structured data.

| Project Name | Project Type | Training Status | Training Dataset |
|---|---|---|---|
| exeML-bb23 | Object Detection | Successful | /obs-modelartwhc/Mc |
| exeML-ee9b | Image Classificat... | Successful | /obs-124c/FLOWER/ |

**Figure 1-21 Creating a predictive analytics project**



* Billing Mode: Pay-per-use

* Name: exeML-d093

* Training Data: delart_datasets/autolearning/bank/train.csv

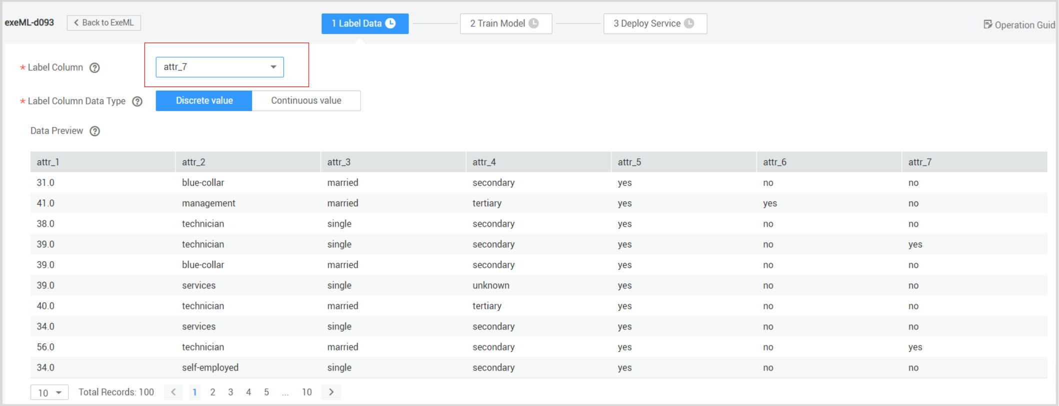
Upload the required data to the OBS bucket in advance. Learn how to upload data to OBS.

Description

0/256

**Figure 1-22 Selecting the data path**

Step 2 Click the project name to enter its **Label Data** page, preview the data and select the training objective (specified by **Label Column**). The training objective here is to determine whether the customer will apply for a deposit (that is, **attr_7**). Then, set **Label Column Data Type** to **Discrete value**. Click **Training**.



* Label Column: attr_7

* Label Column Data Type: Discrete value | Continuous value

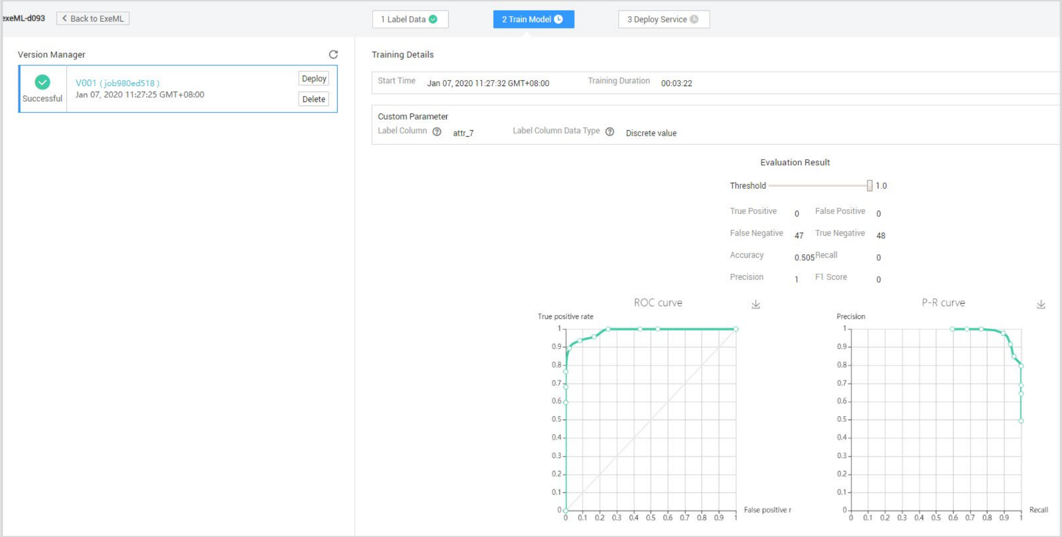
Data Preview

| attr_1 | attr_2 | attr_3 | attr_4 | attr_5 | attr_6 | attr_7 |
|---|---|---|---|---|---|---|
| 31.0 | blue-collar | married | secondary | yes | no | no |
| 41.0 | management | married | tertiary | yes | yes | no |
| 38.0 | technician | single | secondary | yes | no | no |
| 39.0 | technician | single | secondary | yes | no | yes |
| 39.0 | blue-collar | married | secondary | yes | no | no |
| 39.0 | services | single | unknown | yes | no | no |
| 40.0 | technician | married | tertiary | yes | no | no |
| 34.0 | services | single | secondary | yes | no | no |
| 56.0 | technician | married | secondary | yes | no | yes |
| 34.0 | self-employed | single | secondary | yes | no | no |

10 Total Records: 100 < 1 2 3 4 5 ... 10 >

**Figure 1-23 Training job parameters**

Step 3 Wait until the training is completed and view the training result. You can check the training effect of the model based on the evaluation result.



Successful V001 ( job980ed518 ) Jan 07, 2020 11:27:25 GMT+08:00

Training Details

Start Time Jan 07, 2020 11:27:32 GMT+08:00 Training Duration 00:03:22

Custom Parameter

Label Column attr_7 Label Column Data Type Discrete value

Evaluation Result

Threshold 1.0

| | | | |
|---|---|---|---|
| True Positive | 0 | False Positive | 0 |
| False Negative | 47 | True Negative | 48 |
| Accuracy | 0.505 | Recall | 0 |
| Precision | 1 | F1 Score | 0 |

**Figure 1-24 Model training management page**

## 1.4.4.3 Deploying a Service and Performing Prediction

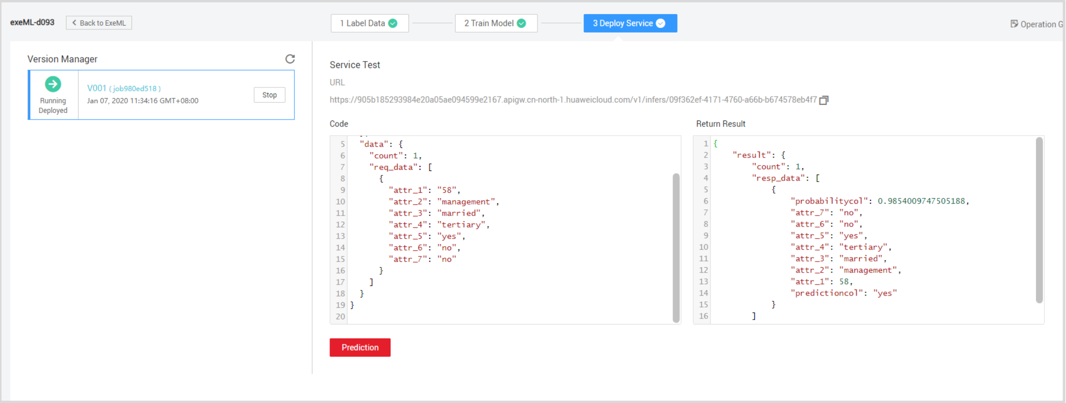
After the training job is completed, you can deploy the trained model as a prediction service as follows.

Step 1 On the Train Model tab page, click Deploy in the upper left corner.

Step 2 On the Deploy Service page, test the prediction service.

Step 3 Use the following code for prediction. You only need to modify the parameters under the req_data module.

```
{
  "meta": {
    "uuid": "10eb0091-887f-4839-9929-cbc884f1e20e"
  },
  "data": {
    "count": 1,
     "req_data": [
       {
         "attr_1": "58",
         "attr_2": "management",
         "attr_3": "married",
         "attr_4": "tertiary",
                                                    "attr_5": "yes",
         "attr_6": "no",
         "attr_7": "no"
       }
     ]
   }
}
```



https://905b185293984e20a05ae094599e2167.apigw.cn-north-1.huaweicloud.com/v1/infers/09f362ef-4171-4760-a66b-b674578eb4f7

Code

```
  "data": {
    "count": 1,
    "req_data": [
      {
        "attr_1": "58",
        "attr_2": "management",
        "attr_3": "married",
        "attr_4": "tertiary",
        "attr_5": "yes",
        "attr_6": "no",
        "attr_7": "no"
      }
    ]
  }
}
```

Return Result

```
{
    "result": {
        "count": 1,
        "resp_data": [
            {
                "probabilitycol": 0.9854009747505188,
                "attr_7": "no",
                "attr_6": "no",
                "attr_5": "yes",
                "attr_4": "tertiary",
                "attr_3": "married",
                "attr_2": "management",
                "attr_1": 58,
                "predictioncol": "yes"
            }
        ]
```

**Figure 1-25 Prediction test result**

# 2 Data Management

## 2.1 About This Lab

The **Data Management** module of ModelArts allows you to upload data, label data, create datasets, and manage data versions. This section mainly describes these functions. The dataset files uploaded in this section will be used for subsequent custom algorithm experiments. Labeling jobs completed in **Data Management** can also be used by training jobs, but labeling jobs created in **ExeML** can be used only by **ExeML**. **Data Management** and **ExeML** use the same labeling techniques.

## 2.2 Objectives

Learn how to use OBS Browser to upload data.

Learn how to create datasets.

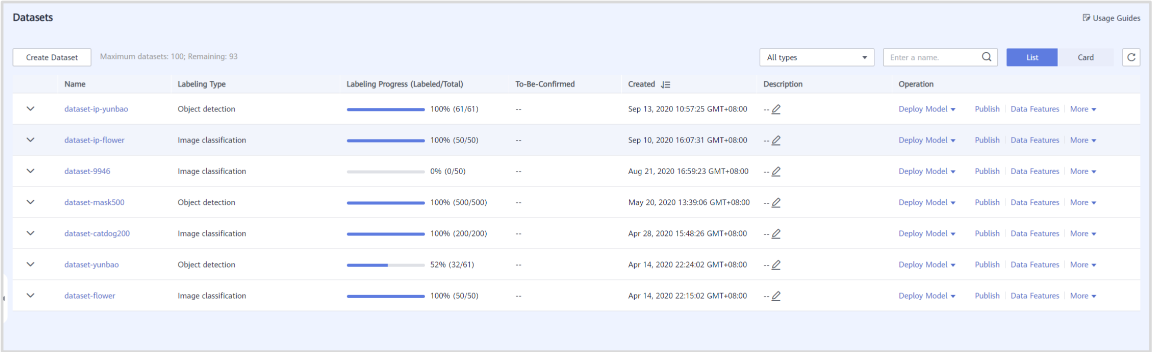
## 2.3 Procedure

### 2.3.1 Data Labeling for Flower Recognition

#### 2.3.1.1 Creating dataset

Step 1 Learn the layout of the **Datasets** page.

The **Datasets** page lists all dataset. On this page, you can click **Create Dataset** to create a dataset, or enter a dataset name in the search box in the upper right corner of the dataset list and click 🔍 to search for a dataset. See Figure 2-1.



Datasets

Usage Guides

Create Dataset Maximum datasets: 100; Remaining: 93

All types ▾ | Enter a name. 🔍 | List | Card

| Name | Labeling Type | Labeling Progress (Labeled/Total) | To-Be-Confirmed | Created | Description | Operation |
|---|---|---|---|---|---|---|
| dataset-ip-yunbao | Object detection | 100% (61/61) | -- | Sep 13, 2020 10:57:25 GMT+08:00 | -- | Deploy Model ▾ Publish \| Data Features \| More ▾ |
| dataset-ip-flower | Image classification | 100% (50/50) | -- | Sep 10, 2020 16:07:31 GMT+08:00 | -- | Deploy Model ▾ Publish \| Data Features \| More ▾ |
| dataset-9946 | Image classification | 0% (0/50) | -- | Aug 21, 2020 16:59:23 GMT+08:00 | -- | Deploy Model ▾ Publish \| Data Features \| More ▾ |
| dataset-mask500 | Object detection | 100% (500/500) | -- | May 20, 2020 13:39:06 GMT+08:00 | -- | Deploy Model ▾ Publish \| Data Features \| More ▾ |
| dataset-catdog200 | Image classification | 100% (200/200) | -- | Apr 28, 2020 15:48:26 GMT+08:00 | -- | Deploy Model ▾ Publish \| Data Features \| More ▾ |
| dataset-yunbao | Object detection | 52% (32/61) | -- | Apr 14, 2020 22:24:02 GMT+08:00 | -- | Deploy Model ▾ Publish \| Data Features \| More ▾ |
| dataset-flower | Image classification | 100% (50/50) | -- | Apr 14, 2020 22:15:02 GMT+08:00 | -- | Deploy Model ▾ Publish \| Data Features \| More ▾ |

**Figure 2-1 Dataset page**

Parameters:

**Name**: name of a data labeling job. After you click the name, the job details page is displayed.

**Labeling Type**: type of a data labeling job. Currently, labeling types include image classification, object detection, sound classification, text classification, and text labeling.

**Labeling Progress**: labeling progress of a data labeling job, displaying also the total number of images and the number of labeled images.

**Created**: time when a data labeling job was created.

**Description**: brief description of a data labeling job.

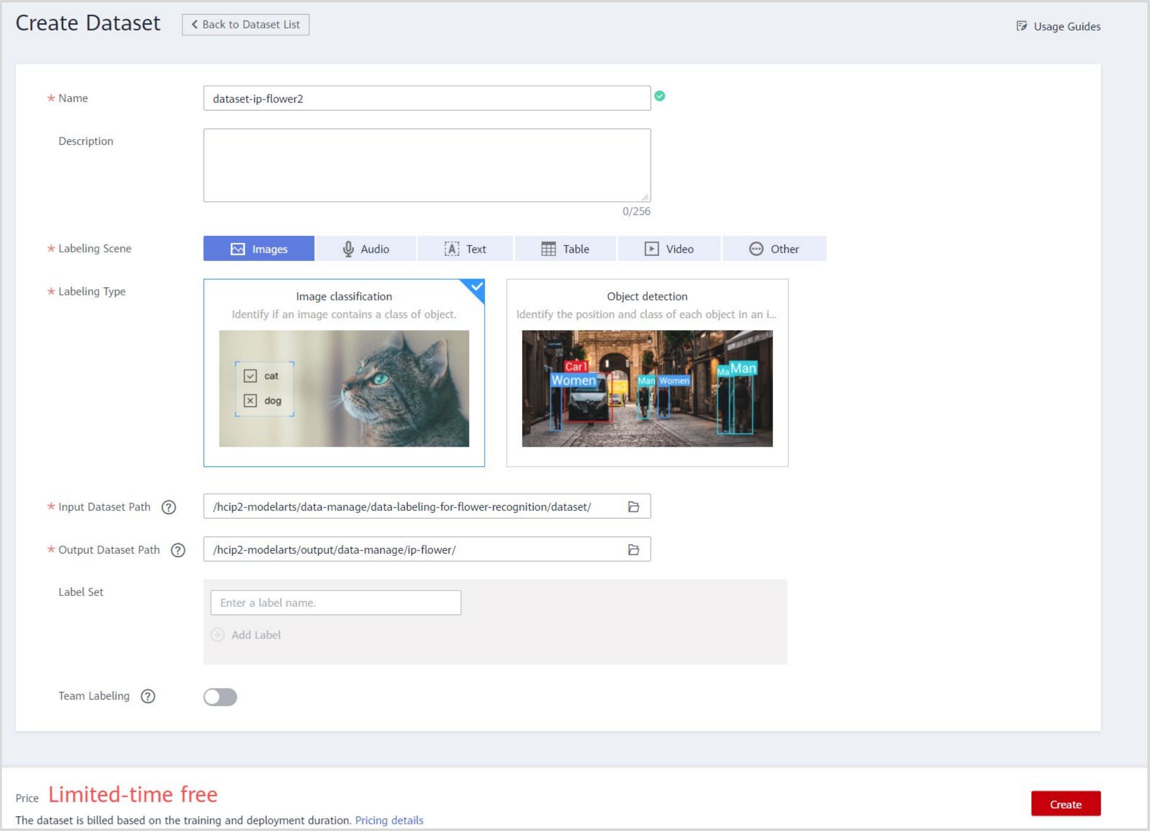
**Operation**: operations you can perform on a data labeling job, including:

- Publish: Publish dataset versions.
- Deploy Model: Deploy the dataset with algorithm.

Step 2 Create a dataset.

On OBS, create an empty folder (obs://hcip2-modelarts/data-manage/data-labeling-for-flower-recognition/dataset/) to store images to be labeled, and create another empty folder (obs://hcip2-modelarts/output/data-manage/ip-flower/) to store the labeling result.

On ModelArts, click **Create Dataset** in the upper left corner of the **Datasets** page. Set required parameters. Then, click **Create**. See Figure 2-2.



Create Dataset

* Name: dataset-ip-flower2

Description: 0/256

* Labeling Scene: Images

* Labeling Type: Image classification — Identify if an image contains a class of object.

* Input Dataset Path: /hcip2-modelarts/data-manage/data-labeling-for-flower-recognition/dataset/

* Output Dataset Path: /hcip2-modelarts/output/data-manage/ip-flower/

Label Set: Enter a label name.

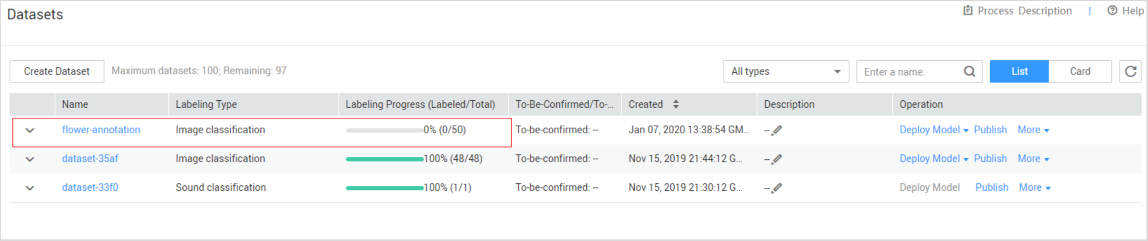
Team Labeling

Price Limited-time free

The dataset is billed based on the training and deployment duration. Pricing details

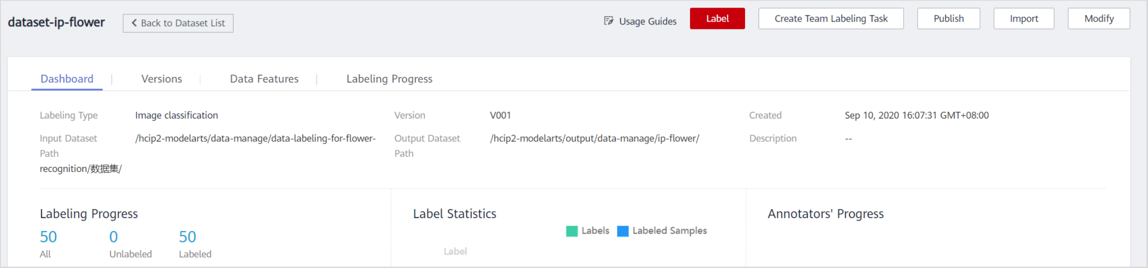
**Figure 2-2 Parameter settings**

After the job is created, click the job name to enter its details page.


Datasets
Process Description | Help
Create Dataset
Maximum datasets: 100; Remaining: 97
All types
Enter a name.
List
Card

| Name | Labeling Type | Labeling Progress (Labeled/Total) | To-Be-Confirmed/To-... | Created | Description | Operation |
|---|---|---|---|---|---|---|
| flower-annotation | Image classification | 0% (0/50) | To-be-confirmed: -- | Jan 07, 2020 13:38:54 GM... | -- | Deploy Model Publish More |
| dataset-35af | Image classification | 100% (48/48) | To-be-confirmed: -- | Nov 15, 2019 21:44:12 G... | -- | Deploy Model Publish More |
| dataset-33f0 | Sound classification | 100% (1/1) | To-be-confirmed: -- | Nov 15, 2019 21:30:12 G... | -- | Deploy Model Publish More |

**Figure 2-3 Datasets page**



| | | | | | |
|---|---|---|---|---|---|
| Labeling Type | Image classification | Version | V001 | Created | Sep 10, 2020 16:07:31 GMT+08:00 |
| Input Dataset Path | /hcip2-modelarts/data-manage/data-labeling-for-flower-recognition/数据集/ | Output Dataset Path | /hcip2-modelarts/output/data-manage/ip-flower/ | Description | -- |

**Figure 2-4 Datasets page**

The images have been uploaded to OBS, click ⟳ to synchronize the images to ModelArts. For details, see Step 1 in section 1.4.2.2 "Labeling Data."

## 2.3.1.2 Labeling Images

For details, see Step 2 in section 1.4.2.2 "Labeling Data."

## 2.3.1.3 Deleting or Modifying a Label in One Image

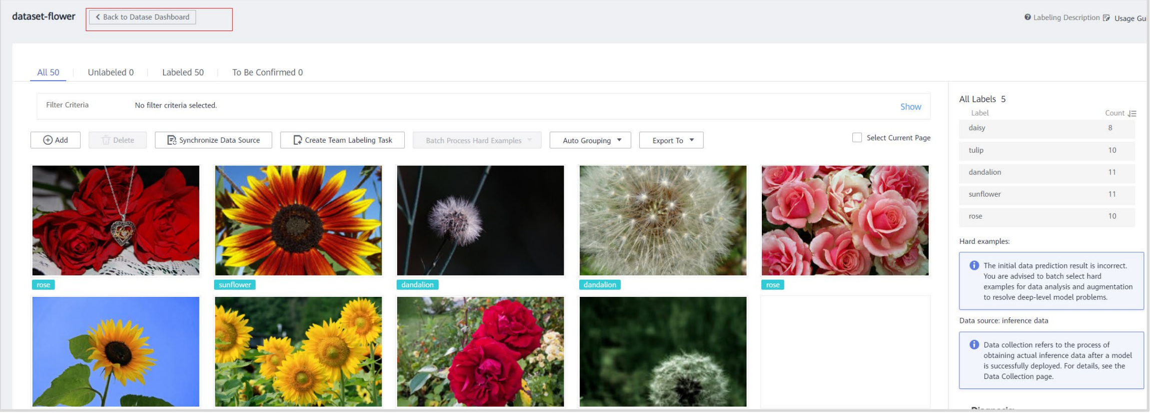
For details, see Step 3 in section 1.4.2.2 "Labeling Data."

## 2.3.1.4 Deleting or Modifying a Label in Multiple Images

For details, see Step 4 in section 1.4.2.2 "Labeling Data."

## 2.3.1.5 Publish a Dataset

After the labeling is complete, return to the Dataset Overview page.



| Label | Count |
|---|---|
| daisy | 8 |
| tulip | 10 |
| dandalion | 11 |
| sunflower | 11 |
| rose | 10 |

Hard examples:

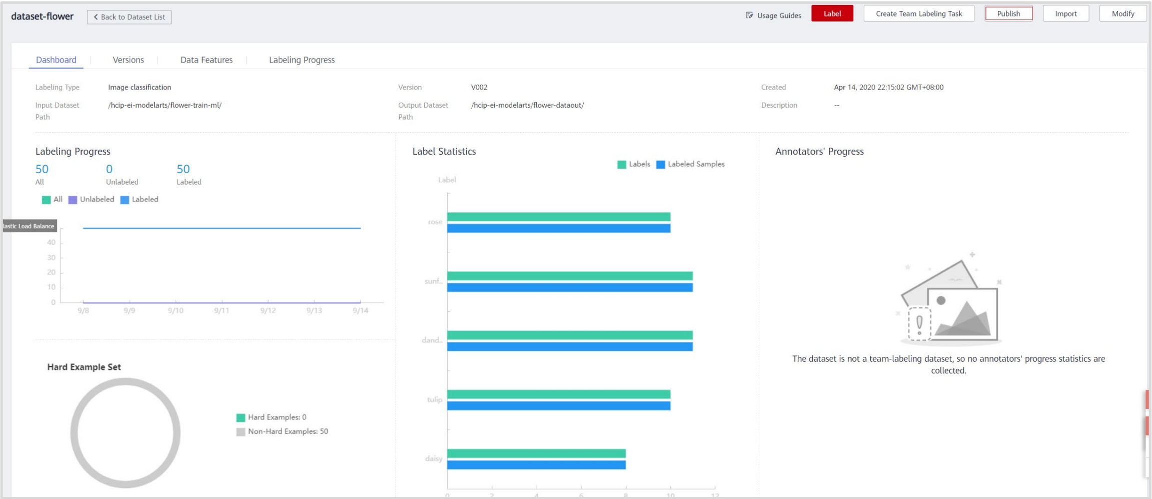
The initial data prediction result is incorrect. You are advised to batch select hard examples for data analysis and augmentation to resolve deep-level model problems.

Data source: inference data

Data collection refers to the process of obtaining actual inference data after a model is successfully deployed. For details, see the Data Collection page.

**Figure 2-5 Labeled**

Click Publish on the labeling page. The dataset is automatically generated. See the following figure. The published dataset can be directly used in training jobs.



Labeling Type Image classification

Input Dataset Path /hcip-ei-modelarts/flower-train-ml/

Version V002

Output Dataset Path /hcip-ei-modelarts/flower-dataout/

Created Apr 14, 2020 22:15:02 GMT+08:00

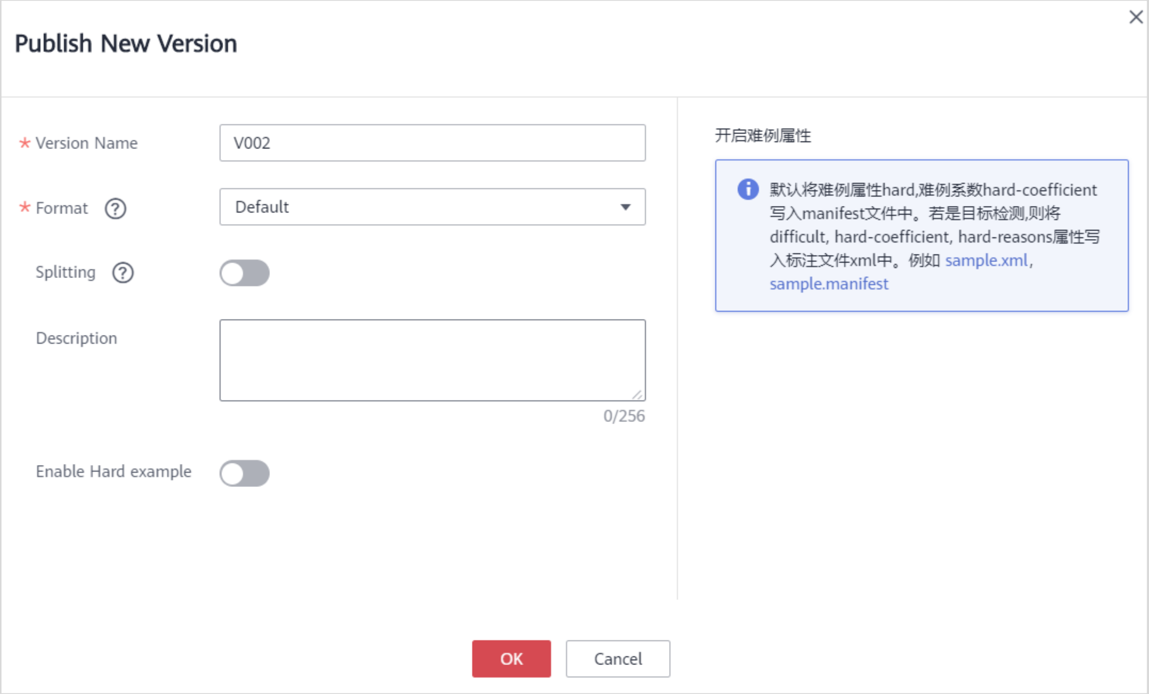
Description --

The dataset is not a team-labeling dataset, so no annotators' progress statistics are collected.

**Figure 2-6 Publish dataset**

## 2.3.1.6 Managing Versioning

Choose **Datasets** > **Version Manager**. On the page that is displayed, you can view the version updates of a dataset. The version name is automatically generated in the form of *v001* After a dataset is created successfully, a temporary version is automatically generated and named in the form of *v001*. To switch the directory, move the cursor to the target version name, and then click **Set to current directory** to set the version to the current directory. The **Add File** and **Delete File** operations in the dataset directory are automatically saved to the temporary version. You can view the number of added and deleted files on the **Version Manager** tab page.



Publish New Version

* Version Name: V002

* Format: Default

Splitting

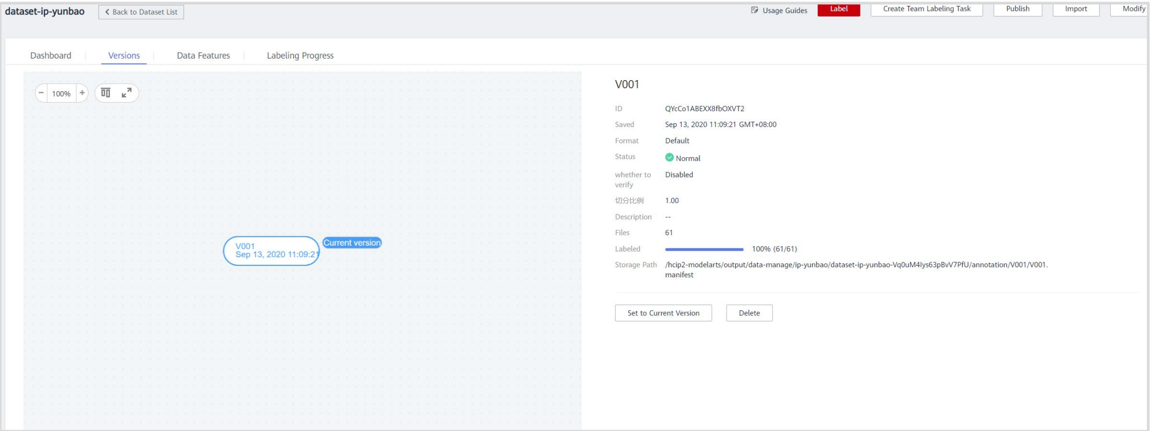
Description: 0/256

Enable Hard example

开启难例属性

默认将难例属性hard,难例系数hard-coefficient写入manifest文件中。若是目标检测,则将difficult, hard-coefficient, hard-reasons属性写入标注文件xml中。例如 sample.xml, sample.manifest

**Figure 2-7** **Publish new version**



dataset-ip-yunbao

V001

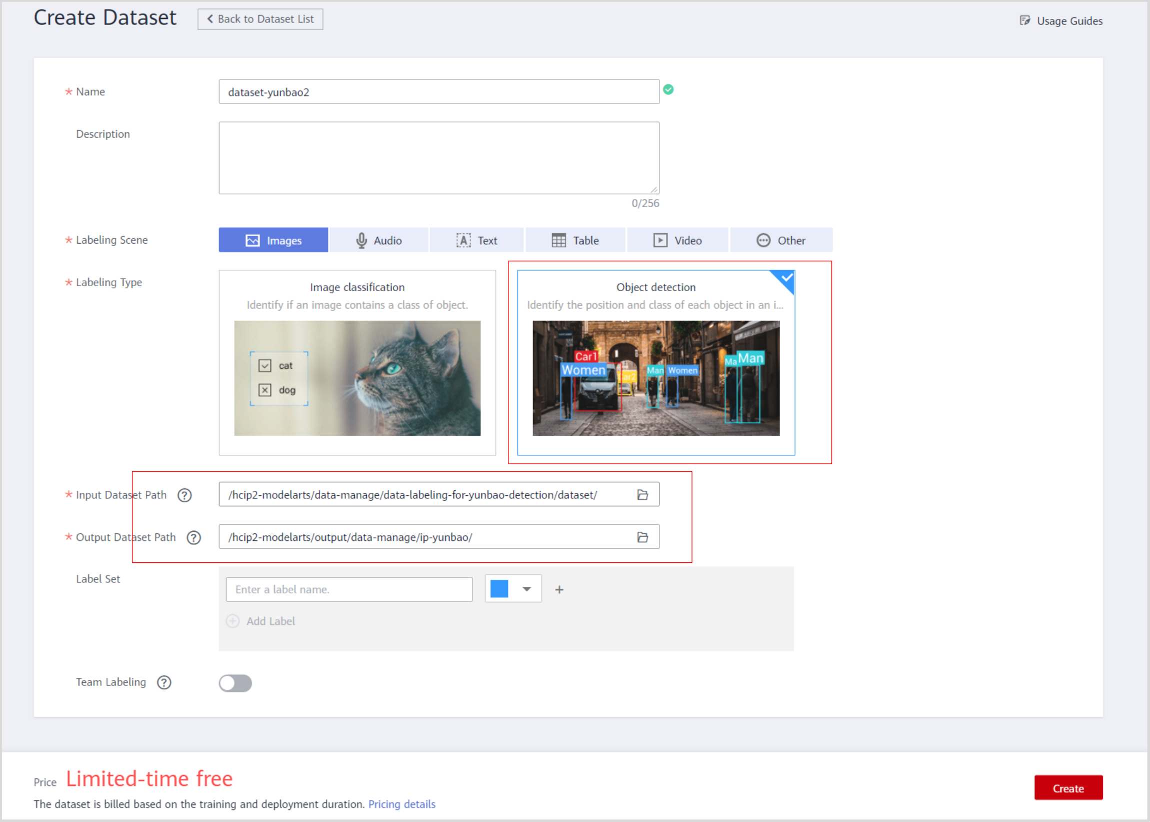
| | |
|---|---|
| ID | QYcCo1ABEXX8fbOXVT2 |
| Saved | Sep 13, 2020 11:09:21 GMT+08:00 |
| Format | Default |
| Status | Normal |
| whether to verify | Disabled |
| 切分比例 | 1.00 |
| Description | -- |
| Files | 61 |
| Labeled | 100% (61/61) |
| Storage Path | /hcip2-modelarts/output/data-manage/ip-yunbao/dataset-ip-yunbao-Vq0uM4lys63pBvV7PfU/annotation/V001/V001.manifest |

**Figure 2-8** **Version managment**

## 2.3.2 Data Labeling for Yunbao Detection

Step 1 Create a dataset.

Log in to ModelArts and click **Create Dataset**. The Create Dataset page is displayed, as shown in the following figure. After setting the parameters, click **Create**.



Create Dataset

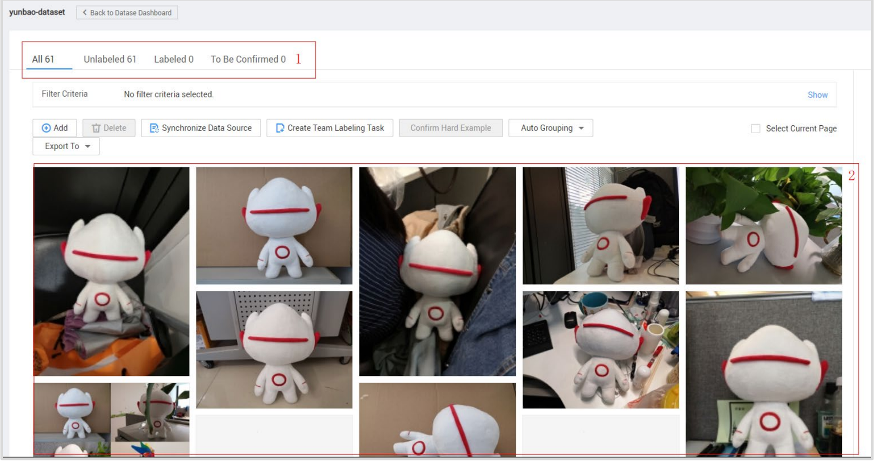
* Name: dataset-yunbao2
Description:
0/256
* Labeling Scene: Images | Audio | Text | Table | Video | Other
* Labeling Type: Image classification — Identify if an image contains a class of object. | Object detection — Identify the position and class of each object in an i...
* Input Dataset Path: /hcip2-modelarts/data-manage/data-labeling-for-yunbao-detection/dataset/
* Output Dataset Path: /hcip2-modelarts/output/data-manage/ip-yunbao/
Label Set: Enter a label name.
Team Labeling

Price Limited-time free
The dataset is billed based on the training and deployment duration. Pricing details

Figure 2-9 Creating dataset

Step 2 Label the data.

After the data labeling job is created, return to the job list and click the job name to enter the labeling page. Upload the image dataset from **modelarts-datasets-and-source-code/data-management/data-labeling-for-yunbao-detection** to this page and label the images. The data is synchronized to the OBS path of the data labeling job by default. Alternatively, you can import the images to OBS and click **Synchronize Data Source** to synchronize them to ModelArts for labeling.



yunbao-dataset

All 61 Unlabeled 61 Labeled 0 To Be Confirmed 0

Filter Criteria: No filter criteria selected.

**Figure 2-10 Data labeling page of an object detection project**

Click the **Unlabeled** tab in area 1, and then click an image in area 2. Then, frame out the object in the image with a labeling box. Ensure that the box does not contain too much background information. Input a label and press **Enter**. See Figure 2-11.

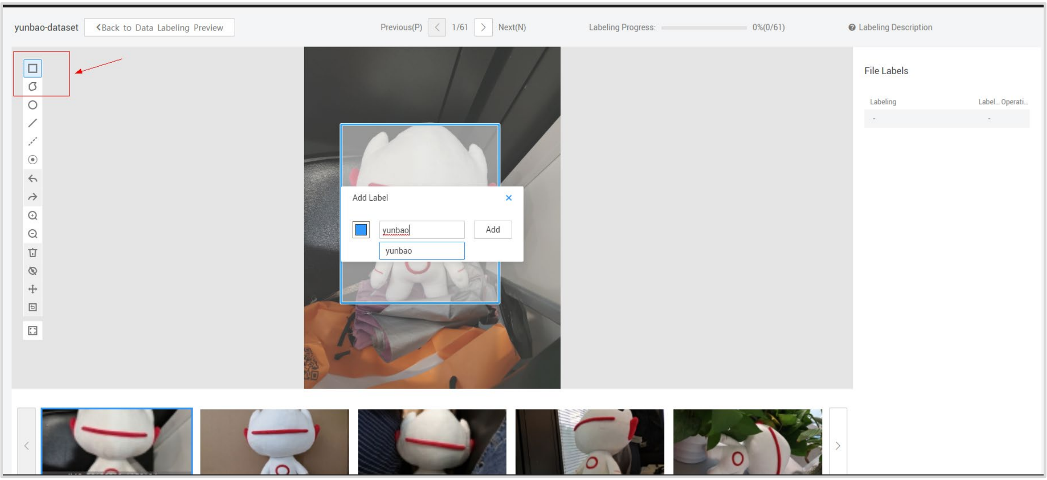


**Figure 2-11 Image labeling for object detection**

Step 3 Delete or modify a label in one image.

In area 1, click the **Labeled** tab and click the target image to enter its labeling page. Then, you can delete or modify a label through either of the following methods:

Method 1: Move the cursor to the labeling box, right-click, and choose **Delete** from the shortcut menu to delete the label, or choose **Modify**, enter a new label name, and press **Enter**.

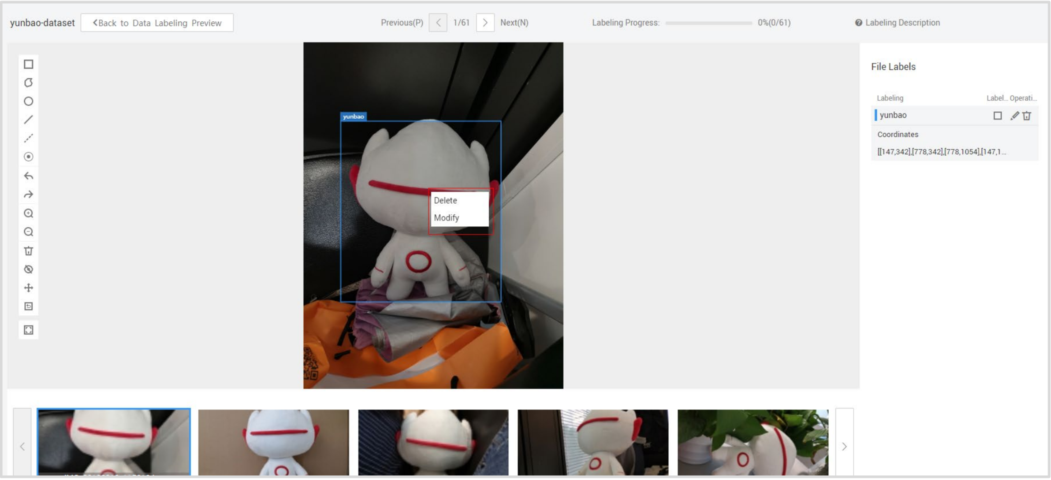


**Figure 2-12 Deleting/Modifying a label in one image**

Method 2: Click the ✎ or 🗑 button on the right of the image to modify or delete its label.

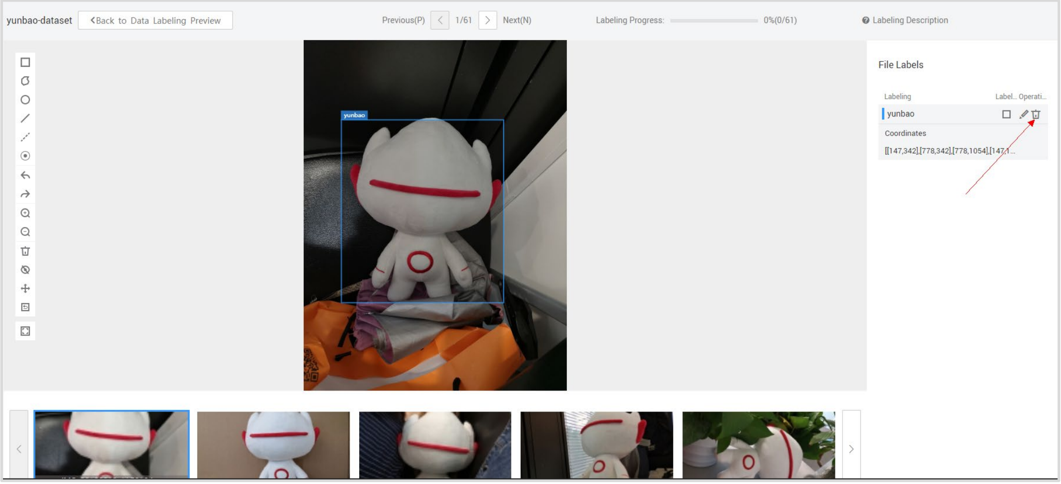


Figure 2-13 Deleting a label and adding a new label in one image

Step 4 Delete or modify a label in multiple images.

On the **Labeled** tab page, click ✎ on the right of the target label to rename it, or click 🗑 to delete it from multiple images. In the dialog box that is displayed, select **Delete label** or **Delete label and images that only contain this label**. See Figure 2-14.

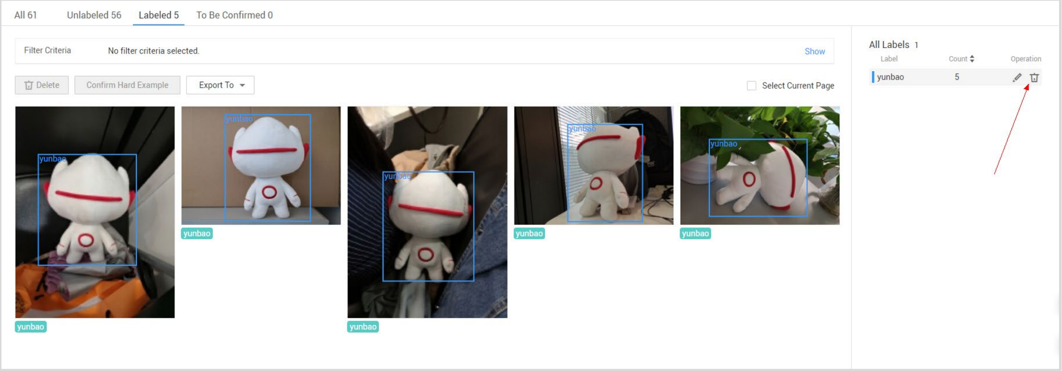
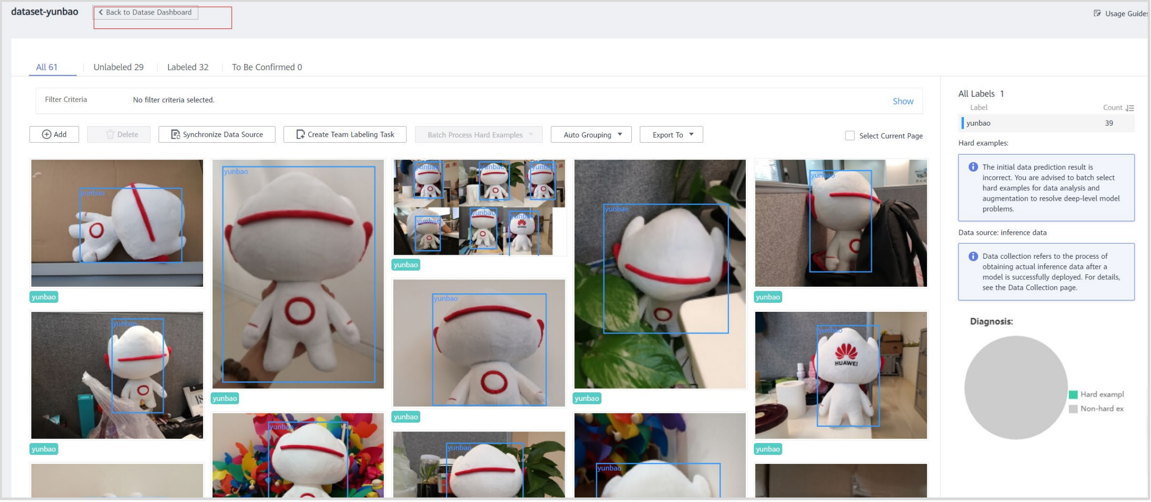


Figure 2-14 Deleting/Modifying a label in multiple images

Step 5 Publish a dataset.

After data labeling is complete, click Back to Dataset Overview on the labeling page.



Filter Criteria: No filter criteria selected.

All Labels 1

| Label | Count |
|---|---|
| yunbao | 39 |

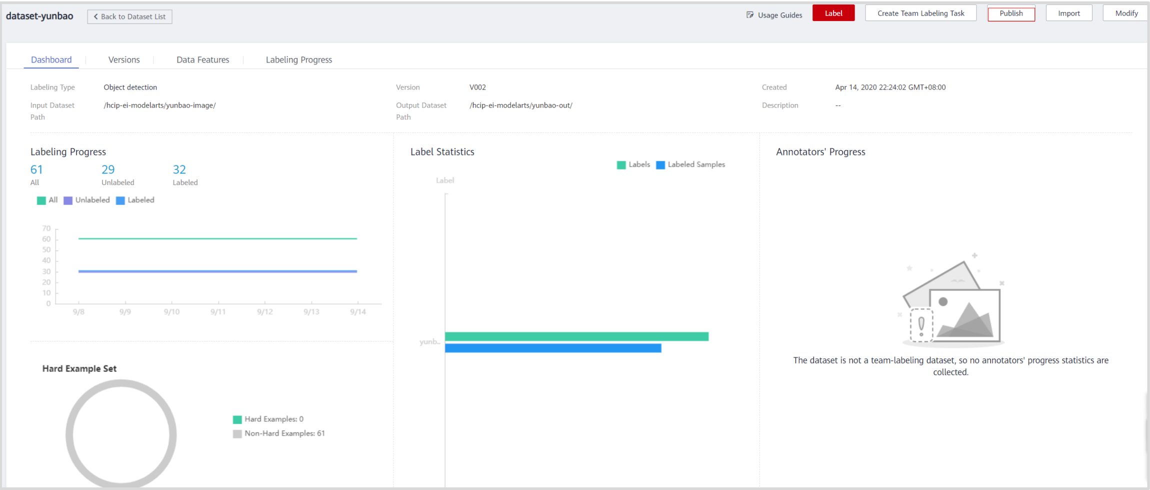
Hard examples:

The initial data prediction result is incorrect. You are advised to batch select hard examples for data analysis and augmentation to resolve deep-level model problems.

Data source: inference data

Data collection refers to the process of obtaining actual inference data after a model is successfully deployed. For details, see the Data Collection page.

Figure 2-15 Labeled



Labeling Type: Object detection

Input Dataset Path: /hcip-ei-modelarts/yunbao-image/

Version: V002

Output Dataset Path: /hcip-ei-modelarts/yunbao-out/

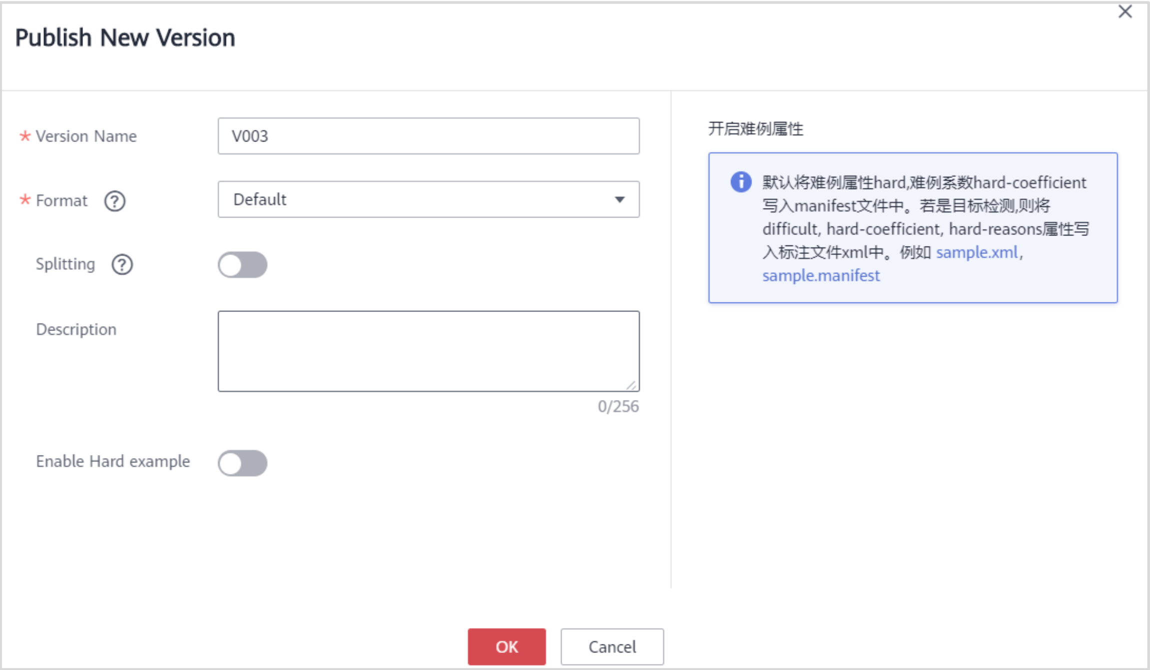
Created: Apr 14, 2020 22:24:02 GMT+08:00

Description: --

The dataset is not a team-labeling dataset, so no annotators' progress statistics are collected.

Figure 2-16 Publish a dataset

Click **OK** to publish.



Publish New Version

* Version Name: V003

* Format: Default

Splitting

Description

0/256

Enable Hard example

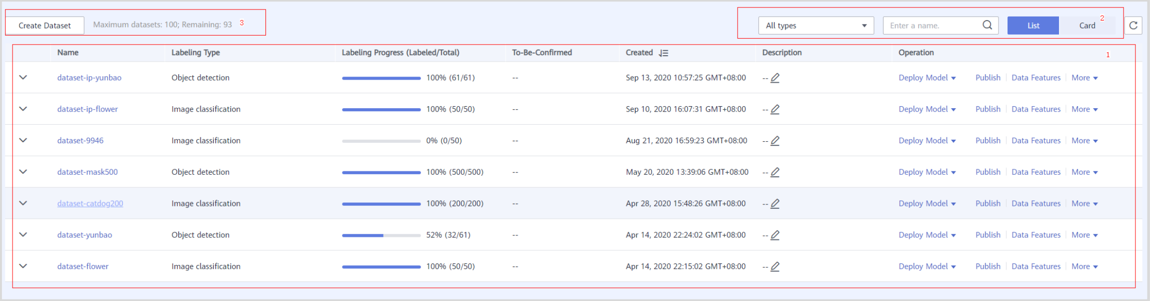
开启难例属性

默认将难例属性hard,难例系数hard-coefficient写入manifest文件中。若是目标检测,则将difficult, hard-coefficient, hard-reasons属性写入标注文件xml中。例如 sample.xml, sample.manifest

Figure 2-17 Publish

Step 6 Manage the dataset.

After creating a dataset, you can manage it on ModelArts.



Maximum datasets: 100; Remaining: 93

|  | Name | Labeling Type | Labeling Progress (Labeled/Total) | To-Be-Confirmed | Created | Description | Operation |
|---|---|---|---|---|---|---|---|
|  | dataset-ip-yunbao | Object detection | 100% (61/61) | -- | Sep 13, 2020 10:57:25 GMT+08:00 | -- | Deploy Model Publish Data Features More |
|  | dataset-ip-flower | Image classification | 100% (50/50) | -- | Sep 10, 2020 16:07:31 GMT+08:00 | -- | Deploy Model Publish Data Features More |
|  | dataset-9946 | Image classification | 0% (0/50) | -- | Aug 21, 2020 16:59:23 GMT+08:00 | -- | Deploy Model Publish Data Features More |
|  | dataset-mask500 | Object detection | 100% (500/500) | -- | May 20, 2020 13:39:06 GMT+08:00 | -- | Deploy Model Publish Data Features More |
|  | dataset-catdog200 | Image classification | 100% (200/200) | -- | Apr 28, 2020 15:48:26 GMT+08:00 | -- | Deploy Model Publish Data Features More |
|  | dataset-yunbao | Object detection | 52% (32/61) | -- | Apr 14, 2020 22:24:02 GMT+08:00 | -- | Deploy Model Publish Data Features More |
|  | dataset-flower | Image classification | 100% (50/50) | -- | Apr 14, 2020 22:15:02 GMT+08:00 | -- | Deploy Model Publish Data Features More |

Figure 2-18 Datasets page

Area 1: dataset list. All operations performed on datasets are displayed in this area. For example:

-- Release: Click Release to release the new dataset.

-- Online: Deploy the dataset as an online task.

-- Delete: Move the cursor to the dataset and click Delete.

Area 2: Query information in the dataset list of the current year.

Area 3: Creating a datasetManage versioning.

Step 7 Version management

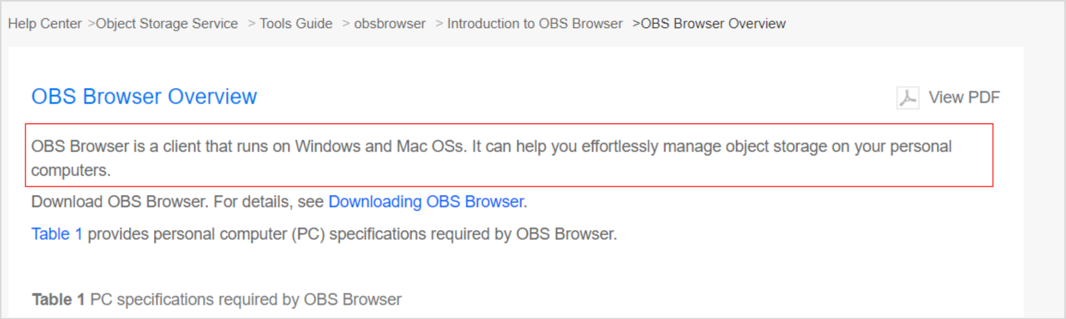
For details, see section 2.3.1.6.

## 2.3.3 Uploading an MNIST Dataset to OBS

Prepare an MNIST dataset, and store it in the **modelarts-datasets-and-source-code/data-management/uploading-a-mnist-dataset-to-obs** directory. Then, upload the prepared dataset to OBS. This experiment describes how to use OBS Browser to upload data to OBS in batches.

Step 1 Obtain the AK/SK. For details, see section 1.3 "Experiment Environment Overview."

Step 2 Download OBS Browser at https://storage.huaweicloud.com/obs/?region=cn-north-1#/obs/buckets. Select a proper version based on your operating system. See Figure 2-19.



Help Center >Object Storage Service > Tools Guide > obsbrowser > Introduction to OBS Browser >OBS Browser Overview

OBS Browser Overview

View PDF

OBS Browser is a client that runs on Windows and Mac OSs. It can help you effortlessly manage object storage on your personal computers.

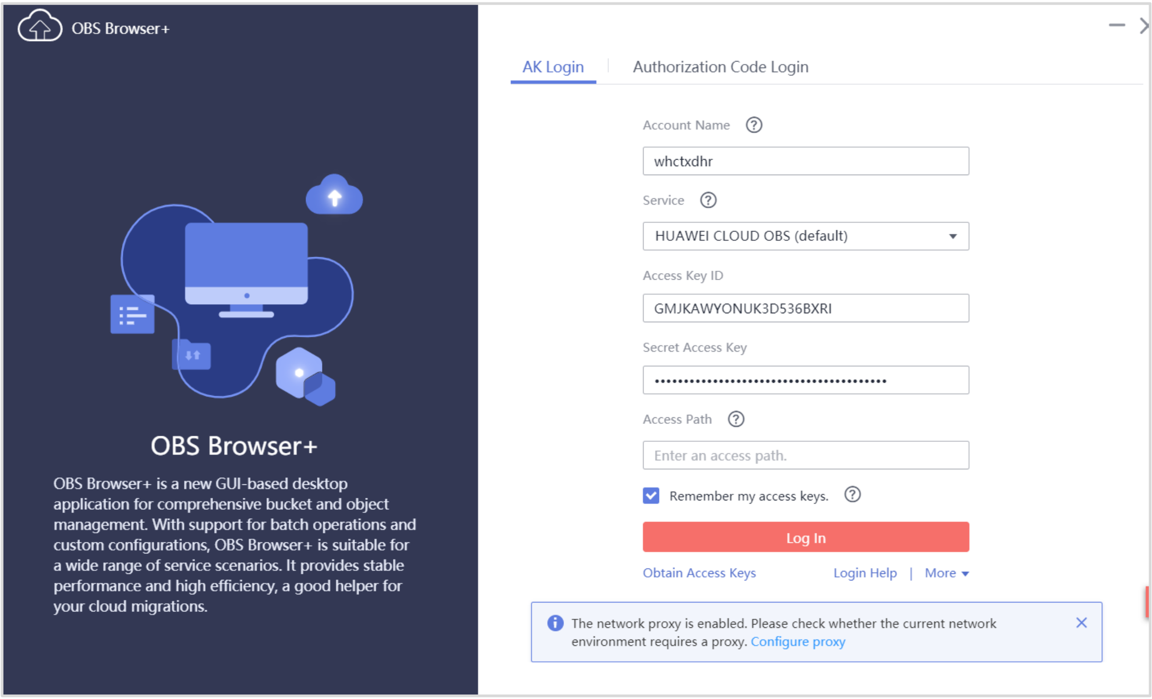
Download OBS Browser. For details, see Downloading OBS Browser.

Table 1 provides personal computer (PC) specifications required by OBS Browser.

**Table 1** PC specifications required by OBS Browser

**Figure 2-19 Downloading OBS Browser**

Decompress the downloaded package and double-click **obs.exe** to open OBS Browser.



OBS Browser+

AK Login | Authorization Code Login

Account Name: whctxdhr

Service: HUAWEI CLOUD OBS (default)

Access Key ID: GMJKAWYONUK3D536BXRI

Secret Access Key: ••••••••••••••••••••••••••••••••••••••••

Access Path: Enter an access path.

☑ Remember my access keys.

Log In

Obtain Access Keys | Login Help | More

The network proxy is enabled. Please check whether the current network environment requires a proxy. Configure proxy

OBS Browser+

OBS Browser+ is a new GUI-based desktop application for comprehensive bucket and object management. With support for batch operations and custom configurations, OBS Browser+ is suitable for a wide range of service scenarios. It provides stable performance and high efficiency, a good helper for your cloud migrations.

**Figure 2-20 Login accounts**

Step 3 Upload files in the MNIST dataset from the modelarts-datasets-and-source-code/data-management/uploading-a-mnist-dataset-to-obs directory to OBS in batches. Wait until the transmission icon in the upper right corner indicates that the uploading is finished. The uploaded dataset can be used in the handwritten digit recognition experiments in sections 4.4.1 and 4.4.3 "Using Native MXNet for Handwritten Digit Recognition."

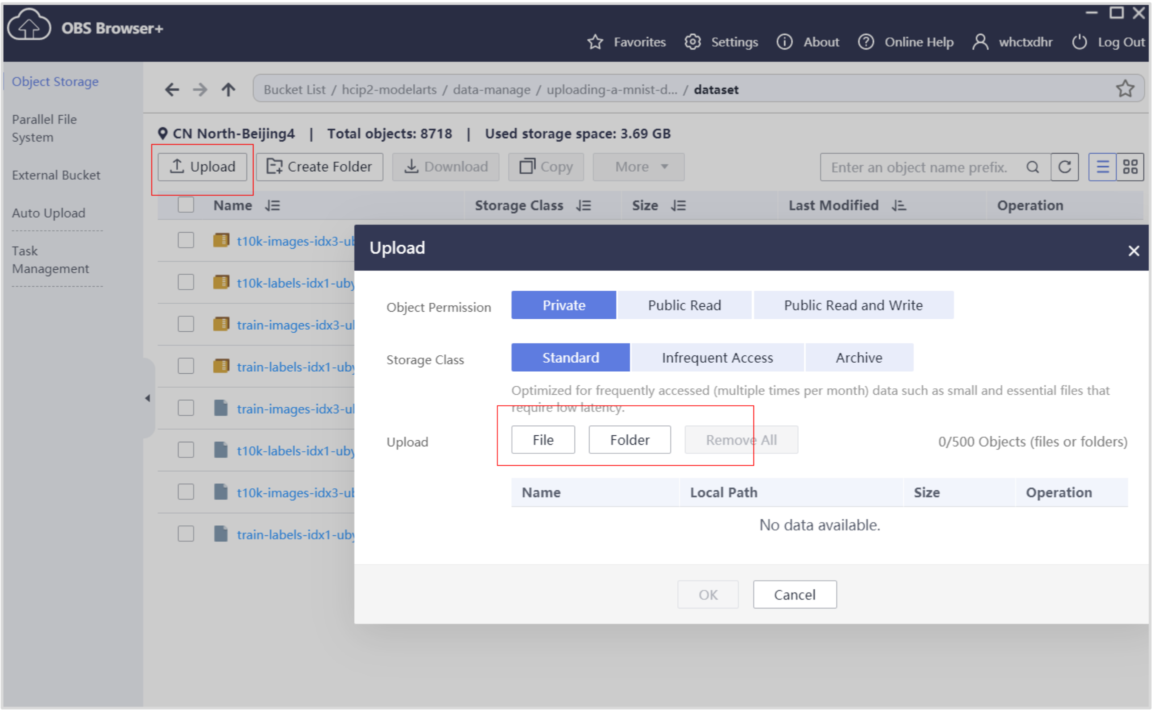
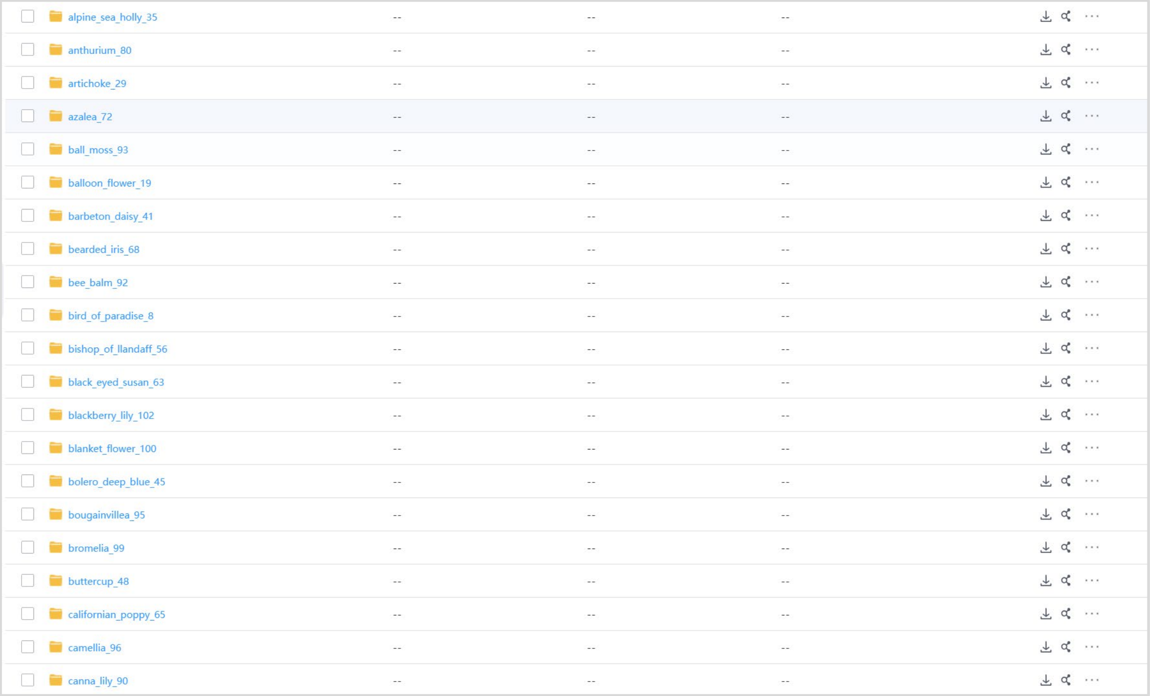


**Figure 2-21 File upload**

## 2.3.4 Uploading of flower classification data set

This dataset will be used for experiments 4.4.2 and 4.4.4 in Chapter 4.

The path of the data set is "ModelArts Experimental Data Set and source code/data management/Flower classification data set upload/data set", under which there are multiple folders with a large number of pictures in each folder.OBS data upload method refer to section 2.3.3.The OBS interface after uploading is as follows:



| | | | | | |
|---|---|---|---|---|---|
| □ | alpine_sea_holly_35 | -- | -- | -- | ··· |
| □ | anthurium_80 | -- | -- | -- | ··· |
| □ | artichoke_29 | -- | -- | -- | ··· |
| □ | azalea_72 | -- | -- | -- | ··· |
| □ | ball_moss_93 | -- | -- | -- | ··· |
| □ | balloon_flower_19 | -- | -- | -- | ··· |
| □ | barbeton_daisy_41 | -- | -- | -- | ··· |
| □ | bearded_iris_68 | -- | -- | -- | ··· |
| □ | bee_balm_92 | -- | -- | -- | ··· |
| □ | bird_of_paradise_8 | -- | -- | -- | ··· |
| □ | bishop_of_llandaff_56 | -- | -- | -- | ··· |
| □ | black_eyed_susan_63 | -- | -- | -- | ··· |
| □ | blackberry_lily_102 | -- | -- | -- | ··· |
| □ | blanket_flower_100 | -- | -- | -- | ··· |
| □ | bolero_deep_blue_45 | -- | -- | -- | ··· |
| □ | bougainvillea_95 | -- | -- | -- | ··· |
| □ | bromelia_99 | -- | -- | -- | ··· |
| □ | buttercup_48 | -- | -- | -- | ··· |
| □ | californian_poppy_65 | -- | -- | -- | ··· |
| □ | camellia_96 | -- | -- | -- | ··· |
| □ | canna_lily_90 | -- | -- | -- | ··· |

**Figure 2-22 File upload**

# 3 Built-in Algorithms for Deep Learning

## 3.1 About This Lab

ModelArts provides a series of built-in models covering image classification and object detection, such as the classic ResNet model and lightweight MobileNet model. Built-in algorithms can greatly shorten the training time on a new dataset and achieve higher precision. Training with built-in algorithms is a common method of deep learning.

## 3.2 Objectives

This lab describes how to use built-in algorithms to train datasets. The process is free of coding, and you only need to prepare datasets that meet specified requirements.

## 3.3 Procedure

### 3.3.1 Flower Recognition Application

This section describes how to use a built-in model on ModelArts to build a flower image classification application. The procedure consists of four parts:

1. Preparing data: On the **Data Management** page of ModelArts, label the images and create a flowers dataset.

2. Training a model: Load a built-in model to train the flowers dataset to generate a new model.

3. Managing a model: Import the new model to manage it.

4. Deploying a model: Deploy the model as a real-time service, batch service, or edge service.

NOTE

If you use ModelArts for the first time, add an access key before using it. For details, see section 1.3 "Experiment Environment Overview."

#### 3.3.1.1 Preparing Data

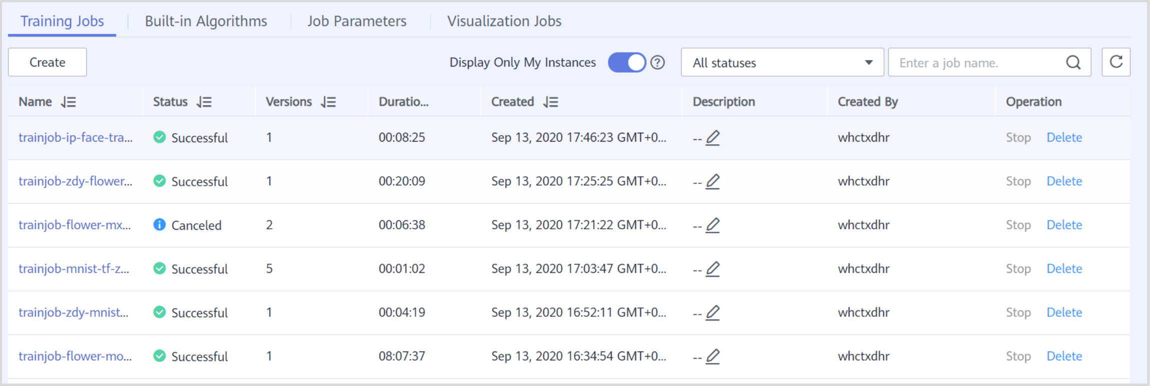
The flower images have been labeled and a dataset version has been created in section 2.3.1 "Data Labeling for Flower Recognition." This experiment uses the labeled flowers dataset.

### 3.3.1.2 Training a Model

On the **Training Jobs** page of ModelArts, you can create training jobs, manage job parameters, and perform operations related to visualization jobs.

The **Training Jobs** page lists all training jobs you created. See Figure 3-1. You can create training jobs, filter the training jobs by status, or search for a training job by entering the job name in the search box.

The following uses the ResNet_v1_50 built-in model as an example to describe how to create a training job and generate a new model.



| Name | Status | Versions | Duratio... | Created | Description | Created By | Operation |
|---|---|---|---|---|---|---|---|
| trainjob-ip-face-tra... | Successful | 1 | 00:08:25 | Sep 13, 2020 17:46:23 GMT+0... | -- | whctxdhr | Stop Delete |
| trainjob-zdy-flower... | Successful | 1 | 00:20:09 | Sep 13, 2020 17:25:25 GMT+0... | -- | whctxdhr | Stop Delete |
| trainjob-flower-mx... | Canceled | 2 | 00:06:38 | Sep 13, 2020 17:21:22 GMT+0... | -- | whctxdhr | Stop Delete |
| trainjob-mnist-tf-z... | Successful | 5 | 00:01:02 | Sep 13, 2020 17:03:47 GMT+0... | -- | whctxdhr | Stop Delete |
| trainjob-zdy-mnist... | Successful | 1 | 00:04:19 | Sep 13, 2020 16:52:11 GMT+0... | -- | whctxdhr | Stop Delete |
| trainjob-flower-mo... | Successful | 1 | 08:07:37 | Sep 13, 2020 16:34:54 GMT+0... | -- | whctxdhr | Stop Delete |

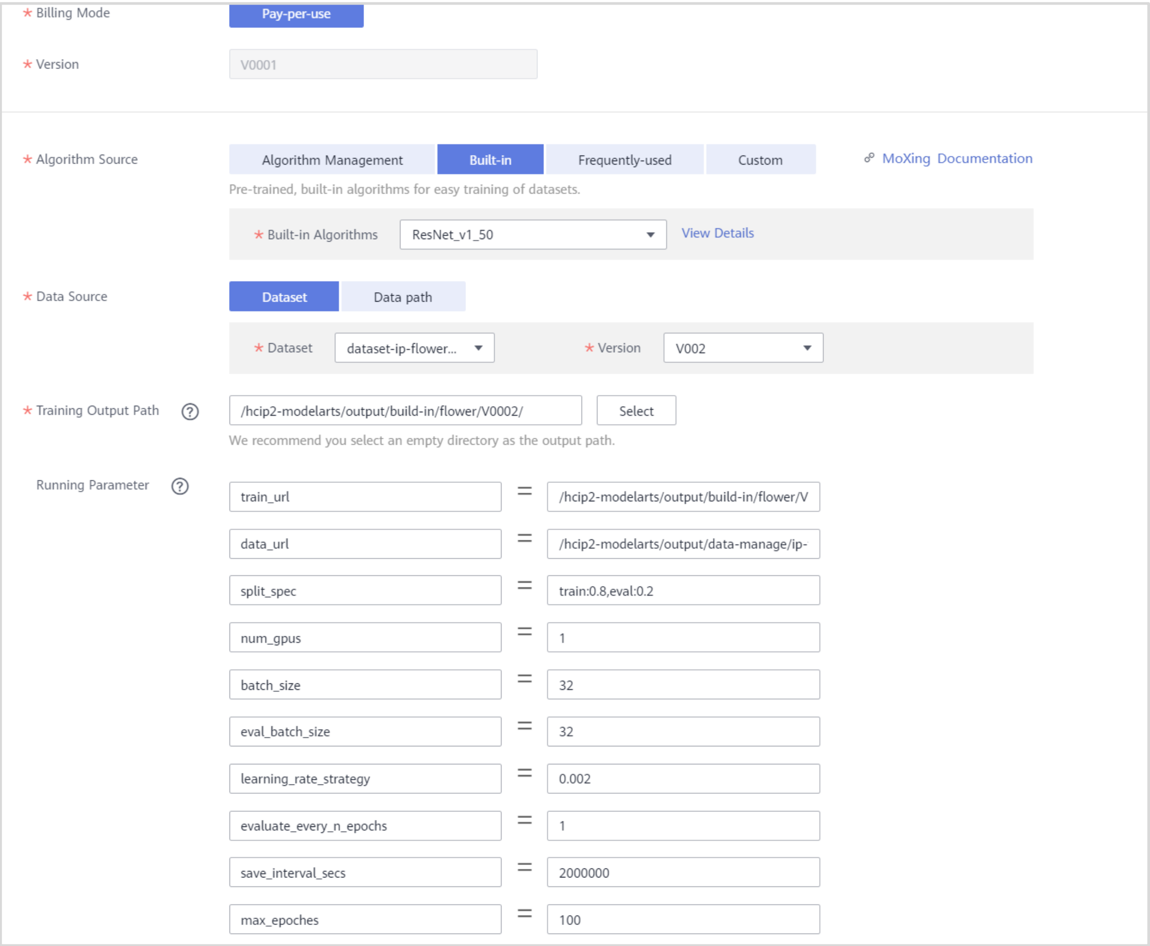
**Figure 3-1 Training Jobs page**

Step 2 Create a training job.

On the ModelArts management console, choose **Training Management > Training Jobs**, and click **Create**. The **Create Training Job** page is displayed.

Step 3 Set required parameters.

On the **Create Training Job** page, set required parameters. Then, click **Next**. After confirming that the configurations are correct, click **Submit**.



* Billing Mode: Pay-per-use

* Version: V0001

* Algorithm Source: Algorithm Management | Built-in | Frequently-used | Custom — MoXing Documentation

Pre-trained, built-in algorithms for easy training of datasets.

* Built-in Algorithms: ResNet_v1_50 — View Details

* Data Source: Dataset | Data path

* Dataset: dataset-ip-flower... * Version: V002

* Training Output Path: /hcip2-modelarts/output/build-in/flower/V0002/ Select

We recommend you select an empty directory as the output path.

Running Parameter

| | | |
|---|---|---|
| train_url | = | /hcip2-modelarts/output/build-in/flower/V |
| data_url | = | /hcip2-modelarts/output/data-manage/ip- |
| split_spec | = | train:0.8,eval:0.2 |
| num_gpus | = | 1 |
| batch_size | = | 32 |
| eval_batch_size | = | 32 |
| learning_rate_strategy | = | 0.002 |
| evaluate_every_n_epochs | = | 1 |
| save_interval_secs | = | 2000000 |
| max_epoches | = | 100 |

**Figure 3-2 Parameter settings**

Parameters:

Billing Mode: Pay-per-use by default.

**Name**: name of a training job. The value can be modified as required.

**Version**: version of a training job. The version number is automatically generated.

**Description**: brief description of a training job.

**Data Source**: data required for training. The options are as follows:

**Dataset**: Select a dataset and its version.

**Data path**: Select the training data from an OBS bucket.

**Algorithm Source**: The options are as follows:

**Built-in**: Select a built-in ModelArts algorithm.

**Frequently-used**: Select an AI engine and its version, the code directory, and the boot file.

**Training Output Path**: This parameter is mandatory. Select the training result storage location to store the output model file. (You need to create an empty OBS folder. In this example, the output path is **/modelarts-demo/builtin-algorithm/output**.)

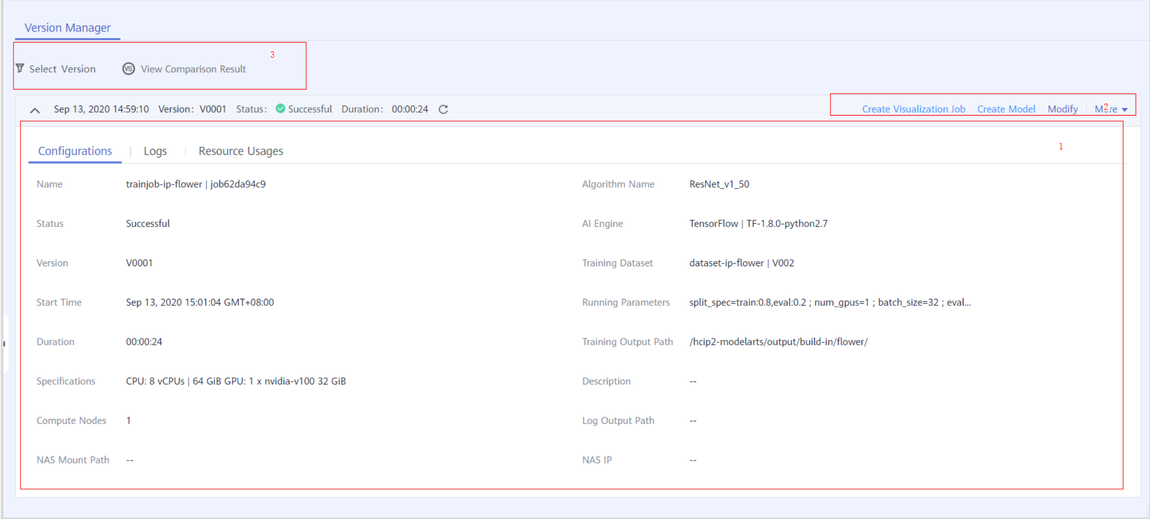
**Job Log Path**: Select a path for storing log files generated during job running.

**Resource Pool**: You must select a resource pool (including CPU and GPU resource pools) for the training job. GPU training is fast while CPU training is slow. GPU/P100 is recommended.

**Compute Nodes**: Specify the number of compute nodes. (One node is used for standalone training, while multiple nodes are used for distributed training. Multi-node distributed training can accelerate the training process.)

Step 4 View the training job.

In the training job list, click the job name to switch to the training job details page. Figure 3-3 shows the **Version Manager** tab page. On the **Traceback Diagrams** tab page, you can view the traceback diagrams of data, training, models, and web services.



Sep 13, 2020 14:59:10 Version: V0001 Status: Successful Duration: 00:00:24

| | | | |
|---|---|---|---|
| Name | trainjob-ip-flower \| job62da94c9 | Algorithm Name | ResNet_v1_50 |
| Status | Successful | AI Engine | TensorFlow \| TF-1.8.0-python2.7 |
| Version | V0001 | Training Dataset | dataset-ip-flower \| V002 |
| Start Time | Sep 13, 2020 15:01:04 GMT+08:00 | Running Parameters | split_spec=train:0.8,eval:0.2 ; num_gpus=1 ; batch_size=32 ; eval... |
| Duration | 00:00:24 | Training Output Path | /hcip2-modelarts/output/build-in/flower/ |
| Specifications | CPU: 8 vCPUs \| 64 GiB GPU: 1 x nvidia-v100 32 GiB | Description | -- |
| Compute Nodes | 1 | Log Output Path | -- |
| NAS Mount Path | -- | NAS IP | -- |

**Figure 3-3 Training job details page**

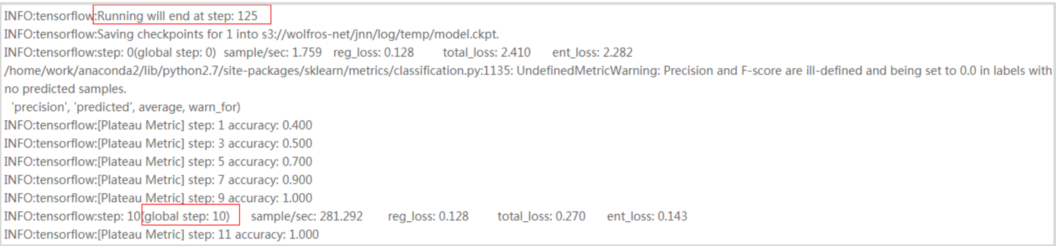
**Area 1**: Displays the details of the current job.

**Area 2**: Create visual jobs and other operations.

**Area 3**: Some operations on the current version.

Step 5 Create a visualization job.

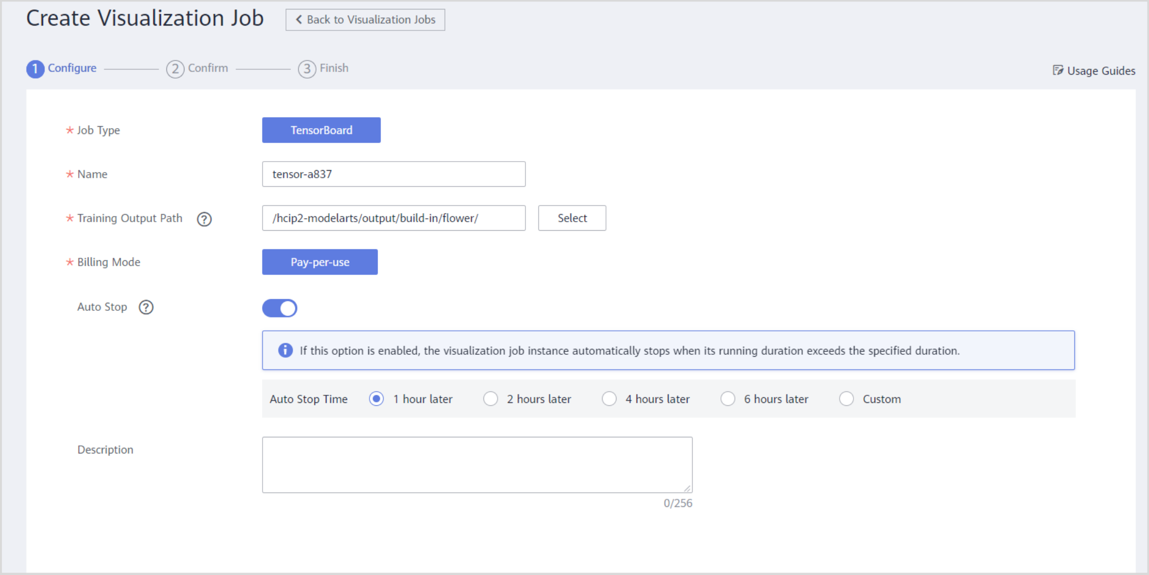
After a training job is created, you can go to its details page to view its log. The log records the current number and the total number of training steps, which can be used as a reference for the training progress. However, if the precision is not significantly improved in a training phase, the training job automatically stops. See Figure 3-4. The log shows that the job will stop after 125 training steps. The current log record shows that 10 training steps have been performed (a training log record is printed every 10 steps by default). If the precision does not increase, the training stops before the number of steps reaches 125.



```
INFO:tensorflow:Running will end at step: 125
INFO:tensorflow:Saving checkpoints for 1 into s3://wolfros-net/jnn/log/temp/model.ckpt.
INFO:tensorflow:step: 0(global step: 0)  sample/sec: 1.759   reg_loss: 0.128       total_loss: 2.410      ent_loss: 2.282
/home/work/anaconda2/lib/python2.7/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with
no predicted samples.
  'precision', 'predicted', average, warn_for)
INFO:tensorflow:[Plateau Metric] step: 1 accuracy: 0.400
INFO:tensorflow:[Plateau Metric] step: 3 accuracy: 0.500
INFO:tensorflow:[Plateau Metric] step: 5 accuracy: 0.700
INFO:tensorflow:[Plateau Metric] step: 7 accuracy: 0.900
INFO:tensorflow:[Plateau Metric] step: 9 accuracy: 1.000
INFO:tensorflow:step: 10(global step: 10)    sample/sec: 281.292      reg_loss: 0.128       total_loss: 0.270      ent_loss: 0.143
INFO:tensorflow:[Plateau Metric] step: 11 accuracy: 1.000
```

Figure 3-4 Training job log

After the training job is completed (its **Status** column on the training job page displays **Successful**), or it has been written to the event file, choose **Training Jobs > Version Manager**, click **Create Visualization Job** in the upper right corner, and enter basic information. See Figure 3-5. You can enter any name. The log path is automatically set to the model storage path, that is, the **Training Output Path** parameter in the training job. Click **Next**. After confirming that the configurations are correct, click **Submit**. You can return to the **Visualization Jobs** page and click the job name to view its details. You need to manually stop the visualization job after using it to avoid additional charges.



* Job Type: TensorBoard

* Name: tensor-a837

* Training Output Path: /hcip2-modelarts/output/build-in/flower/

* Billing Mode: Pay-per-use

Auto Stop: on

If this option is enabled, the visualization job instance automatically stops when its running duration exceeds the specified duration.

Auto Stop Time: ● 1 hour later ○ 2 hours later ○ 4 hours later ○ 6 hours later ○ Custom

Description: 0/256

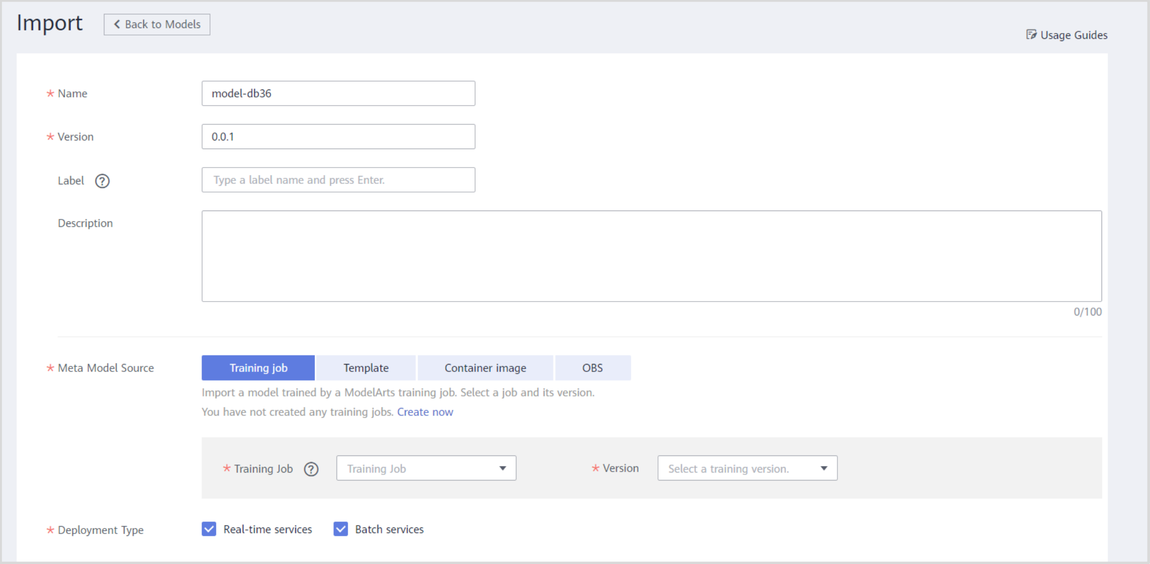
Figure 3-5 Creating a visualization job

### 3.3.1.3 Managing a Model

Step 1 Create a model.

Click the training job name to go to its details page. On the **Version Manager** tab page, click **Create Model** in the upper right corner, enter the model name and version, and click **Next**. The **Models** page is displayed. When the model status becomes **Normal**, the model is successfully created.

Alternatively, click **Import** in the upper left corner of the **Models** page. The **Import** page is displayed. Set required parameters and click **Next** to import a model. See Figure 3-6.



* Name: model-db36

* Version: 0.0.1

Label: Type a label name and press Enter.

Description: 0/100

* Meta Model Source: Training job | Template | Container image | OBS

Import a model trained by a ModelArts training job. Select a job and its version.

You have not created any training jobs. Create now

* Training Job: Training Job ▾ * Version: Select a training version. ▾

* Deployment Type: ☑ Real-time services ☑ Batch services

**Figure 3-6 Importing a model**

Parameters:

**Name**: name of the model.

**Version**: version of the model to be created.

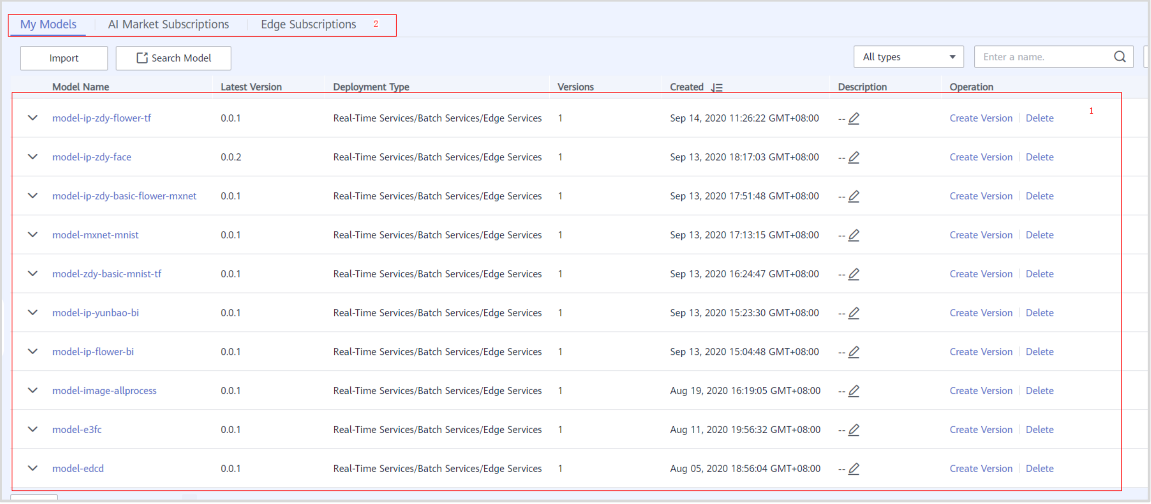
**Description**: brief description of the model.

**Meta Model Source**: You can import a meta model from a training job or OBS.

**Training job**: Select a meta model from a ModelArts training job.

**OBS**: Import a meta model from OBS and select the meta model storage path and AI engine. The meta model imported from OBS must meet the model package specifications.

The following describes the **Model Management** pages:



| Model Name | Latest Version | Deployment Type | Versions | Created | Description | Operation |
|---|---|---|---|---|---|---|
| model-ip-zdy-flower-tf | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 14, 2020 11:26:22 GMT+08:00 | -- | Create Version \| Delete |
| model-ip-zdy-face | 0.0.2 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 13, 2020 18:17:03 GMT+08:00 | -- | Create Version \| Delete |
| model-ip-zdy-basic-flower-mxnet | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 13, 2020 17:51:48 GMT+08:00 | -- | Create Version \| Delete |
| model-mxnet-mnist | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 13, 2020 17:13:15 GMT+08:00 | -- | Create Version \| Delete |
| model-zdy-basic-mnist-tf | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 13, 2020 16:24:47 GMT+08:00 | -- | Create Version \| Delete |
| model-ip-yunbao-bi | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 13, 2020 15:23:30 GMT+08:00 | -- | Create Version \| Delete |
| model-ip-flower-bi | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 13, 2020 15:04:48 GMT+08:00 | -- | Create Version \| Delete |
| model-image-allprocess | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Aug 19, 2020 16:19:05 GMT+08:00 | -- | Create Version \| Delete |
| model-e3fc | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Aug 11, 2020 19:56:32 GMT+08:00 | -- | Create Version \| Delete |
| model-edcd | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Aug 05, 2020 18:56:04 GMT+08:00 | -- | Create Version \| Delete |

**Figure 3-7 Model management pages**

Area 1:

Model list, which lists the models created by users, and the following actions can be taken:

Delete: After selecting the model, click "" on the right side of the model to delete the currently selected model.

Create a new version: Adjust parameters to generate a new version of the model.

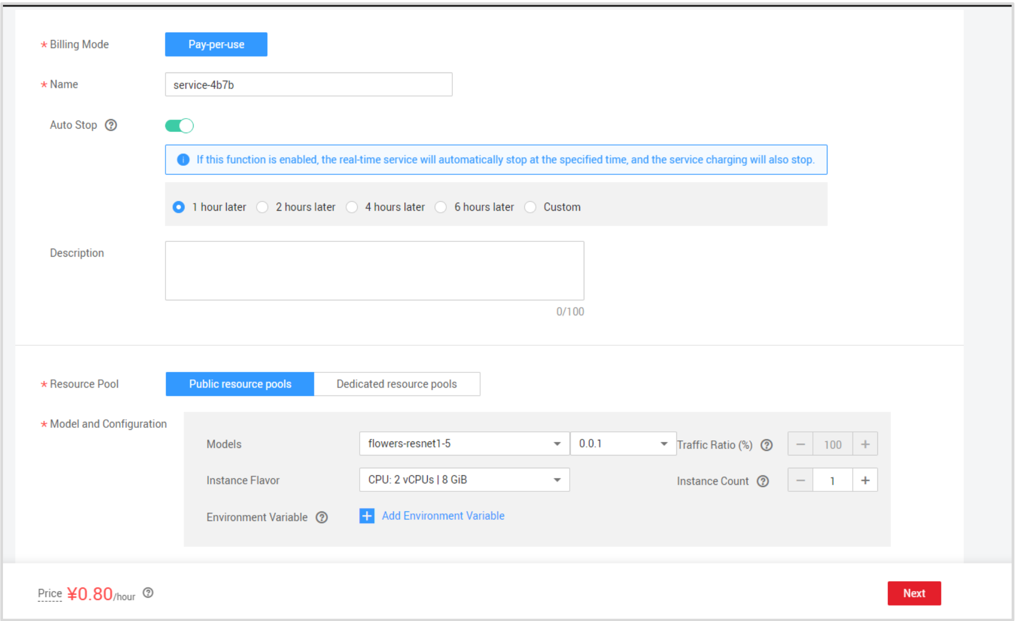
Area 2:

Listed all the current model model information, different access channels, management model.Import and view the relevant models.

### 3.3.1.4 Deploying a Model

After a training job is completed and a model is generated (the model status is **Normal** after being imported), you can deploy the model on the **Service Deployment** page. You can also deploy a model imported from OBS.

Step 1 Click **Deploy** in the upper left corner of the **Real-Time Services** page. On the displayed page, set required parameters. See Figure 3-8. Then, click **Next**. After confirming that the parameter settings are correct, click **Submit** to deploy the real-time service.



* Billing Mode: Pay-per-use

* Name: service-4b7b

Auto Stop

If this function is enabled, the real-time service will automatically stop at the specified time, and the service charging will also stop.

● 1 hour later ○ 2 hours later ○ 4 hours later ○ 6 hours later ○ Custom

Description

0/100

* Resource Pool: Public resource pools | Dedicated resource pools

* Model and Configuration

Models: flowers-resnet1-5 | 0.0.1 | Traffic Ratio (%): 100

Instance Flavor: CPU: 2 vCPUs | 8 GiB | Instance Count: 1

Environment Variable: + Add Environment Variable

Price ¥0.80/hour

Next

**Figure 3-8 Real-time service**

Parameters:

**Name**: name of the real-time service.

**Description**: brief description of the real-time service.

Billing Mode: Pay-per-use

**Models**: Select a model and a version.

**Traffic Ratio**: Set the traffic proportion of the node. If you deploy only one version of a model, set this parameter to **100%**. If you select multiple versions for gray release, ensure that the sum of the traffic ratios of multiple versions is **100%**.

Instance Flavor: Values include 2 vCPUs | 8 GiB and 2 vCPUs | 8 GiB GPU: 1 x P4 and so on.

Instance Count: Select 1 or 2.

**Environment Variable**: Set environment variables.

Step 2 Click the service name to go to its details page. When its status becomes Running, you can debug the code or add an image to test the service. For details about the test operations, see Step 2 in section 1.4.2.4 "Deploying a Service and Performing Prediction." The test image is stored in modelarts-datasets-and-source-code/data-management/built-in-deep-learning-algorithms/flower-recognition-application/test-data. You need to manually stop the real-time service after using it to avoid additional charges.

## 3.3.2 Yunbao Detection Application

This section describes how to use a built-in model on ModelArts to build a Yunbao detection application. The procedure consists of four parts:

1. Preparing data: On the **Data Management** page of ModelArts, label the images and create a Yunbao dataset.

2. Training a model: Load a built-in model to train the Yunbao dataset to generate a new model.

3. Deploying a model: Deploy the obtained model as a real-time prediction service.

4. Initiating a prediction request: Initiate a prediction request and obtain the prediction result.

**NOTE**

If you use ModelArts for the first time, add an access key before using it. For details, see section 1.3 "Experiment Environment Overview."

### 3.3.2.1 Preparing Data

The data has been prepared in section 2.3.2 "Data Labeling for Yunbao Detection."

### 3.3.2.2 Training a Model

On the **Training Jobs** page of ModelArts, you can create training jobs, manage job parameters, and perform operations related to visualization jobs.

The **Training Jobs** page lists all training jobs you created. See Figure 3-1. You can create training jobs, filter the training jobs by status, or search for a training job by entering the job name in the search box.

The following uses the Faster_RCNN_ResNet_v1_50 built-in model as an example to describe how to create a training job and generate a new model.

Step 1 Create a training job.

On the ModelArts management console, choose Training Management > Training Jobs, and click Create. The Create Training Job page is displayed.

Step 2 Set required parameters.

On the **Create Training Job** page, set required parameters. See Figure 3-2. Then, click **Next**. After confirming that the configurations are correct, click **Submit**.

Parameters:

Billing Mode: Pay-per-use by default

**Name**: name of a training job. The value can be modified as required.

**Version**: version of a training job. The version number is automatically generated.

**Description**: brief description of a training job.

**Data Source**: data required for training. The options are as follows:

**Dataset**: Select a dataset and its version.

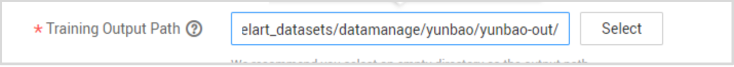
**Data path**: Select the training data from an OBS bucket.

**Algorithm Source**: The options are as follows:

**Built-in**: Select a built-in ModelArts algorithm.

**Frequently-used**: Select an AI engine and its version, the code directory, and the boot file.

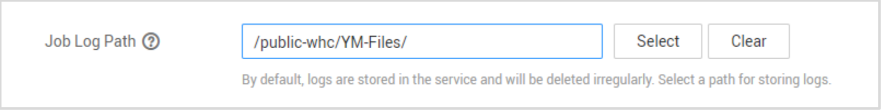
**Training Output Path**: Select a path for storing the training result and save the model file. The path must be empty to ensure normal model training. See Figure 3-9.



* Training Output Path ⓘ elart_datasets/datamanage/yunbao/yunbao-out/

**Figure 3-9 Training output**

**Job Log Path**: Select a path for storing log files generated during job running. This parameter is optional. See Figure 3-10.

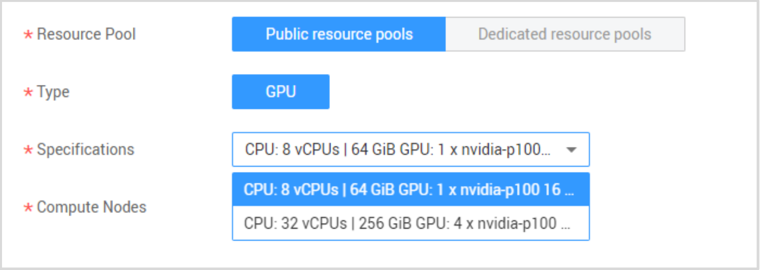


Job Log Path ⓘ /public-whc/YM-Files/

By default, logs are stored in the service and will be deleted irregularly. Select a path for storing logs.

**Figure 3-10 Job log path**

**Resource Pool**: Select a resource pool for the training job. In this example, select the GPU resources. See Figure 3-11.



* Resource Pool: Public resource pools | Dedicated resource pools

* Type: GPU

* Specifications: CPU: 8 vCPUs | 64 GiB GPU: 1 x nvidia-p100...

* Compute Nodes

CPU: 8 vCPUs | 64 GiB GPU: 1 x nvidia-p100 16 ...
CPU: 32 vCPUs | 256 GiB GPU: 4 x nvidia-p100 ...

**Figure 3-11 Resource pool**

**Co**mpute Nodes: Specify the number of compute nodes. Set the value to 1 here.

NOTE

The training takes about 10 minutes if five epochs are running. If the precision is insufficient, increase the number of epochs.

Step 3 View the training job.

In the training job list, click the job name to enter the training job details page.See Step 3 of 3.3.1.2 for details

Step 4 Create a visualization job.

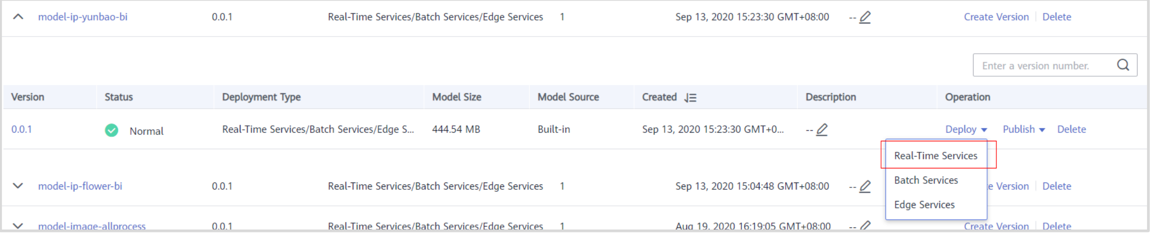
See Step 4 of 3.3.1.2 for details.

Step 5 Create a model.

See Section 3.3.1.3 for details

Step 6 Deploy a real-time service.

When the model status becomes **Normal**, click **Real-Time Services** under **Deploy** to deploy the model as a real-time service. See Figure 3-12.



| | | | | | |
|---|---|---|---|---|---|
| model-ip-yunbao-bi | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 13, 2020 15:23:30 GMT+08:00 | -- | Create Version \| Delete |

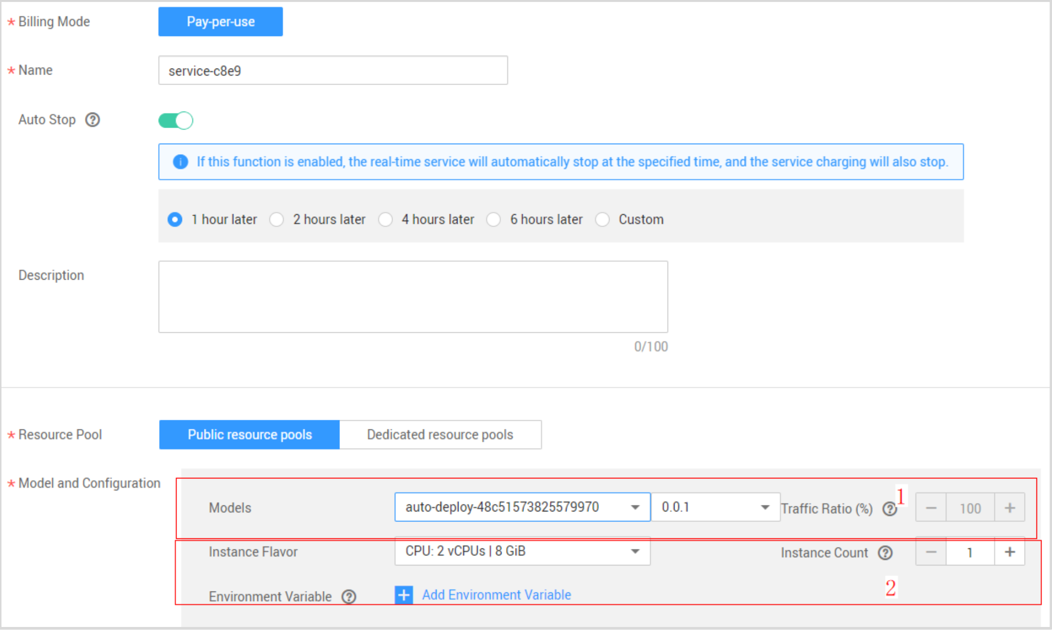
Enter a version number.

| Version | Status | Deployment Type | Model Size | Model Source | Created | Description | Operation |
|---|---|---|---|---|---|---|---|
| 0.0.1 | Normal | Real-Time Services/Batch Services/Edge S... | 444.54 MB | Built-in | Sep 13, 2020 15:23:30 GMT+0... | -- | Deploy \| Publish \| Delete |

| | | | | | |
|---|---|---|---|---|---|
| model-ip-flower-bi | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 13, 2020 15:04:48 GMT+08:00 | -- | ...e Version \| Delete |
| model-image-allprocess | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Aug 19, 2020 16:19:05 GMT+08:00 | -- | Create Version \| Delete |

**Figure 3-12 Service deployment**

**Area 1** displays the version number of the created model, and **area 2** displays the specifications of the selected inference and prediction node. By default, a single CPU node is selected.



* Billing Mode: Pay-per-use

* Name: service-c8e9

Auto Stop ⓘ: on

If this function is enabled, the real-time service will automatically stop at the specified time, and the service charging will also stop.

● 1 hour later ○ 2 hours later ○ 4 hours later ○ 6 hours later ○ Custom

Description: 0/100

* Resource Pool: Public resource pools | Dedicated resource pools

* Model and Configuration

Models: auto-deploy-48c51573825579970 | 0.0.1 | Traffic Ratio (%) ⓘ 100

Instance Flavor: CPU: 2 vCPUs | 8 GiB | Instance Count ⓘ 1

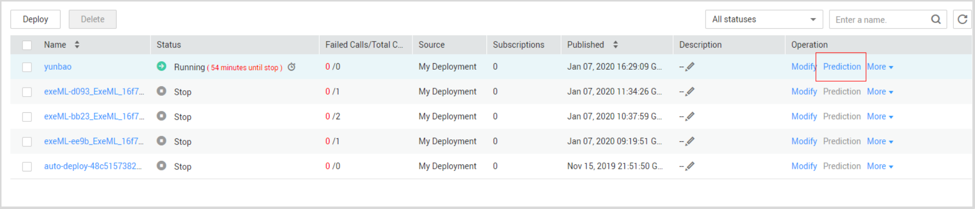
Environment Variable ⓘ: + Add Environment Variable

Figure 3-13 Deployment procedure

After the real-time service is deployed and runs properly, you can perform prediction. After the experiment, you need to manually stop it to stop the billing.

Step 7 Verify the service online.

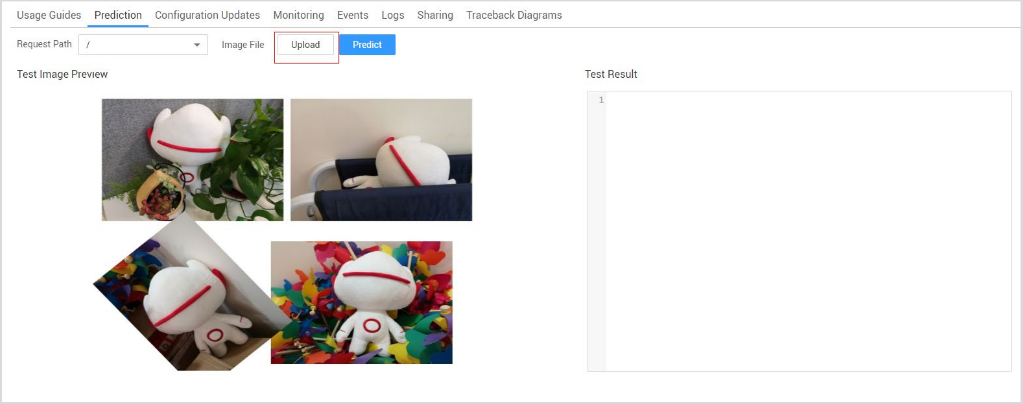
Choose **Service Deployment > Real-Time Services**, and click the deployed real-time service to enter its page.



| □ | Name | Status | Failed Calls/Total C... | Source | Subscriptions | Published | Description | Operation |
|---|---|---|---|---|---|---|---|---|
| □ | yunbao | Running ( 54 minutes until stop ) | 0 /0 | My Deployment | 0 | Jan 07, 2020 16:29:09 G... | -- | Modify Prediction More |
| □ | exeML-d093_ExeML_16f7... | Stop | 0 /1 | My Deployment | 0 | Jan 07, 2020 11:34:26 G... | -- | Modify Prediction More |
| □ | exeML-bb23_ExeML_16f7... | Stop | 0 /2 | My Deployment | 0 | Jan 07, 2020 10:37:59 G... | -- | Modify Prediction More |
| □ | exeML-ee9b_ExeML_16f7... | Stop | 0 /1 | My Deployment | 0 | Jan 07, 2020 09:19:51 G... | -- | Modify Prediction More |
| □ | auto-deploy-48c5157382... | Stop | 0 /0 | My Deployment | 0 | Nov 15, 2019 21:51:50 G... | -- | Modify Prediction More |

Figure 3-14 Entering the service

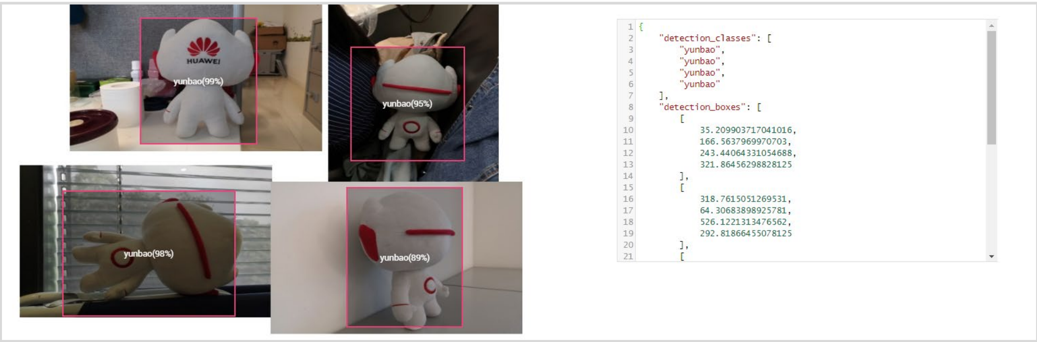
Click the Prediction tab, and click Upload to upload an image for predictive analysis. The path of the test image is in the modelarts-datasets-and-source-code/data-management/built-in-deep-learning-algorithms/yunbao-detection-application/test-data.



Request Path /

Figure 3-15 Uploading an image

The following lists the test result:



```
{
    "detection_classes": [
        "yunbao",
        "yunbao",
        "yunbao",
        "yunbao"
    ],
    "detection_boxes": [
        [
            35.209903717041016,
            166.5637969970703,
            243.44064331054688,
            321.86456298828125
        ],
        [
            318.7615051269531,
            64.30683898925781,
            526.1221313476562,
            292.8186645507812 5
        ],
        [
```

Figure 3-16 Test result

# 4 Custom Basic Algorithms for Deep Learning

## 4.1 About This Lab

This section describes how to use custom algorithms to train and deploy models for real-time prediction on the ModelArts platform. Custom algorithms include algorithms developed based on native TensorFlow and MXNet APIs and algorithms developed based on the self-developed MoXing framework. MoXing can effectively lower the threshold for using deep learning engines, such as TensorFlow and MXNet, and improve performance of distributed training.

## 4.2 Objectives

Upon completion of this task, you will be able to:

- Modify native code to adapt to model training, deployment, and prediction on ModelArts.
- Set up a MoXing framework and use MoXing distributed training capabilities to accelerate training.

## 4.3 Using MoXing

MoXing is a network model development API provided by HUAWEI CLOUD ModelArts. Compared with native APIs such as TensorFlow and MXNet, MoXing APIs make model code compilation easier and can automatically obtain high-performance distributed execution capabilities.

The MoXing module includes the following modules, as shown in Figure 4-1.

- Common module framework (import moxing as mox)
- TensorFlow module (import moxing.tensorflow as mox)
- MXNet module (import moxing.mxnet as mox)
- PyTorch module (import moxing.pytorch as mox)

(When you import engine-related modules, common modules will also be imported.)

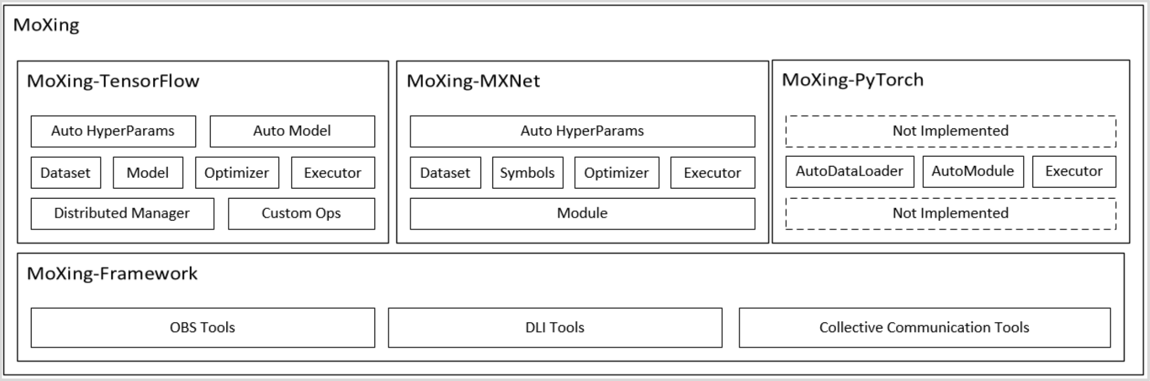


**Figure 4-1 MoXing module**

## 4.3.2 MoXing – Framework Module

You can use the **mox.file** module in MoXing to call APIs to directly access OBS. All environments in ModelArts have been configured.

Example:

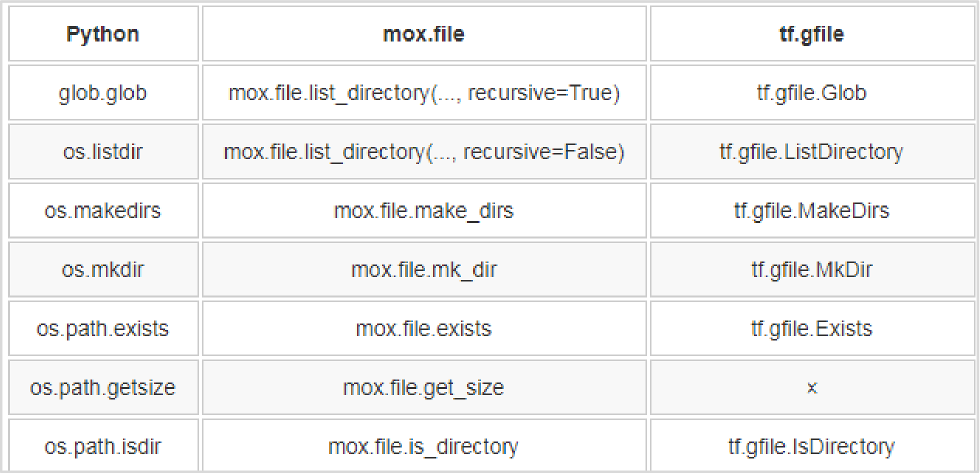
```
import moxing as mox
file_list = mox.file.list_directory('s3://modelarts-demo/codes')
```

In addition to direct access to OBS, you can use the cache directory **/cache** as the transit of OBS in a GPU-enabled job environment, eliminating the need to reconstruct some code for file access.

Example:

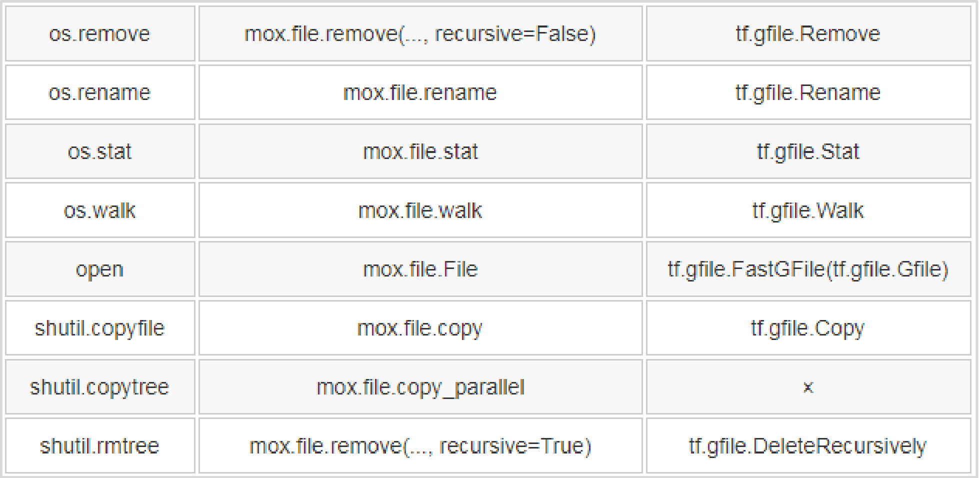
```
import moxing as mox
# Download data from OBS to the local cache.
mox.file.copy_parallel('s3://my_bucket/imput_data', '/cache/input_data')
# Directly use the dataset in the local cache /cache/input_data to start training jobs and save the training output to the local cache /cache/output_log.
train(data_url='/cache/input_data', train_url='/cache/output_log')
# Upload the local cache to OBS.
mox.file.copy_parallel('/cache/output_log', 's3://my_bucket/output_log')
```

API reference:



| Python | mox.file | tf.gfile |
|---|---|---|
| glob.glob | mox.file.list_directory(..., recursive=True) | tf.gfile.Glob |
| os.listdir | mox.file.list_directory(..., recursive=False) | tf.gfile.ListDirectory |
| os.makedirs | mox.file.make_dirs | tf.gfile.MakeDirs |
| os.mkdir | mox.file.mk_dir | tf.gfile.MkDir |
| os.path.exists | mox.file.exists | tf.gfile.Exists |
| os.path.getsize | mox.file.get_size | × |
| os.path.isdir | mox.file.is_directory | tf.gfile.IsDirectory |

**Figure 4-2 APIs (a)**



| | | |
|---|---|---|
| os.remove | mox.file.remove(..., recursive=False) | tf.gfile.Remove |
| os.rename | mox.file.rename | tf.gfile.Rename |
| os.stat | mox.file.stat | tf.gfile.Stat |
| os.walk | mox.file.walk | tf.gfile.Walk |
| open | mox.file.File | tf.gfile.FastGFile(tf.gfile.Gfile) |
| shutil.copyfile | mox.file.copy | tf.gfile.Copy |
| shutil.copytree | mox.file.copy_parallel | × |
| shutil.rmtree | mox.file.remove(..., recursive=True) | tf.gfile.DeleteRecursively |

**Figure 4-3 APIs (b)**

## 4.3.3 MoXing-TensorFlow Module

MoXing-TensorFlow is encapsulated and optimized based on TensorFlow, as shown in Figure 4-4. With the MoXing-TensorFlow programming framework, you only need to pay attention to the implementation of datasets and models. After the standalone training script is implemented, it is automatically extended to distributed training.

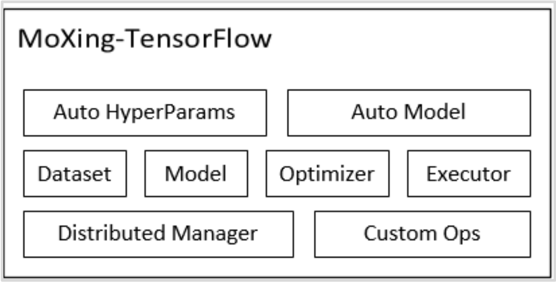


**Figure 4-4 MoXing-TensorFlow optimization**

Dataset: Classification (multilabel), object_detection…

Model: resnet, vgg, inception, mobilenet…

Optimizer: batch_gradients, dynamic_momentum, LARS...

...

MoXing-TensorFlow programming framework:

```
import tensorflow as tf
import moxing as mox
# Define the data input. Receive parameter mode, whose possible values are mox.ModeKeys.TRAIN,
#mox.ModeKeys.EVAL, and mox.ModeKeys.PREDICT. If several tf.Tensor variables are returned,
indicating the input datasets.
def input_fn(mode):
...
  return input_0,input_1,...

# Receive the return value of input_fn as the input. model_fn is used to implement the model and
return a ModelSpec instance.
def model_fn(inputs, mode):
  input_0, input_1 , ... = inputs
  logits, _ = mox.get_model_fn(name='resnet_v1_50',
                                                   run_mode=run_mode,
                                                   ...)
  loss = ...
  return mox.ModelSpec(loss=loss, log_info={'loss': loss}, ...)

# Define an optimization operator. Parameters are not accepted. An optimizer is returned.
def optimizer_fn():
  opt = ...
  return opt
# mox.run defines the entire running process.
mox.run(input_fn=input_fn,
            model_fn=model_fn,
            optimizer_fn=optimizer_fn,
            run_mode=mox.Modekeys.TRAIN,
            ...)
```

NOTE

**mox.ModelSpec**: return value of **model_fn** defined by the user and used to describe a user-defined model.

**loss**: loss value of the user model. The training objective is to decrease the loss value.

**log_info**: monitoring metrics (only scalars) that need to be printed on the console and the visualization job interface during training

**export_spec**: an instance of mox.ExportSpec, which is used to specify the model to be exported.

**hooks**: hooks registered with **tf.Session**

**mox.ExportSpec**: class of the model to be exported

**inputs_dict**: model input node

**outputs_dict**: model output node

**version**: model version

Description of the **mox.run** parameter:

input_fn: user-defined input_fn

model_fn: user-defined model_fn

optimizer_fn: user-defined optimizer_fn

**run_mode**: running mode, **mox.ModeKey.TRAIN**, **mox.ModeKey.EVAL**, ... Only in **TRAIN** mode, the loss gradient descent is performed and parameters are updated.

**log_dir**: destination address of the visualization job log file, checkpoint file, and the exported PB model file

**max_number_of_steps**: maximum number of running steps

**checkpoint_path**: preloaded checkpoint path, which is frequently used in finetuning

**log_every_n_steps**: console print frequency

**save_summary_steps**: visualization job log saving frequency

**save_model_secs**: checkpoint model saving frequency

**export_model**: type of the exported model. Generally, the value is **mox.ExportKeys.TF_SERVING**.

# 4.4 Procedure

## 4.4.1 Using Native TensorFlow for Handwritten Digit Recognition

This section describes how to use custom scripts to train and deploy models for prediction on ModelArts. This section uses TensorFlow as an example to describe how to recognize handwritten digits. The procedure consists of five parts:

Preparing data: Import the MNIST dataset.

Compiling scripts: Use the TensorFlow framework to compile model training scripts.

Training a model: Use the compiled script to train the MNIST dataset to obtain a well-trained model.

Managing a model: Import the model for deployment.

Deploying a model: Deploy the model as a real-time service, batch service, or edge service.

### 4.4.1.1 Preparing Data

You need to prepare data.

### 4.4.1.2 Compiling Scripts

Scripts include training script **train_mnist_tf.py**, inference script **customize_service.py**, and configuration file **config.json**. The inference script and the configuration file are used during model inference, that is, model deployment. Model inference must comply with the following specifications:

Structure of the TensorFlow-based model package

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

│ │ │ ├── variables.data-00000-of-00001

| │ ├── **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 **.py** file on which **customize_service.py** depends can be directly put in the **model** directory.

Step 1 Interpret code.

Training code overview: Training code uses the native TensorFlow code to train the MNIST dataset, that is, to process a task that classifies images to 10 categories. Each image contains 28 x 28 pixels. The network structure is a simple linear model. Initialization of all parameters is zero and training starts from scratch.

The following is training code. The source code is stored in the following path: modelarts-datasets-and-source-code/custom-basic-algorithms-for-deep learning/native-TensorFlow-for-handwritten-digit-recognition/code/train_mnist_tf.py

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import sys

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
# Maximum number of model training steps
tf.flags.DEFINE_integer('max_steps', 1000, 'number of training iterations.')
# Model export version
tf.flags.DEFINE_integer('model_version', 1, 'version number of the model.')
# data_url indicates the data storage path of the data source on the GUI. It is a path of s3://.
tf.flags.DEFINE_string('data_url', '/home/jnn/nfs/mnist', 'dataset directory.')
# File output path, that is, the training output path displayed on the GUI. It is also a path of s3://.
tf.flags.DEFINE_string('train_url', '/home/jnn/temp/delete', 'saved model directory.')

FLAGS = tf.flags.FLAGS

def main(*args):
  # Train the model.
  print('Training model...')
  # Read the MNIST dataset.
```

```
    mnist = input_data.read_data_sets(FLAGS.data_url, one_hot=True)
    sess = tf.InteractiveSession()
    # Create input parameters.
    serialized_tf_example = tf.placeholder(tf.string, name='tf_example')
    feature_configs = {'x': tf.FixedLenFeature(shape=[784], dtype=tf.float32),}
    tf_example = tf.parse_example(serialized_tf_example, feature_configs)
    x = tf.identity(tf_example['x'], name='x')
    y_ = tf.placeholder('float', shape=[None, 10])
    # Create training parameters.
    w = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))
    # Initialize parameters.
    sess.run(tf.global_variables_initializer())
    # Use only the simple linear network layer and define the network output layer softmax.
    y = tf.nn.softmax(tf.matmul(x, w) + b, name='y')
    # Define the loss function.
    cross_entropy = -tf.reduce_sum(y_ * tf.log(y))
    # Add summary information.
    tf.summary.scalar('cross_entropy', cross_entropy)

    # Define the optimizer.
    train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
    # Obtain the accuracy.
    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))
    tf.summary.scalar('accuracy', accuracy)
    # Summarize summary information.
    merged = tf.summary.merge_all()
    # Write data to the summary file every second.
    test_writer = tf.summary.FileWriter(FLAGS.train_url, flush_secs=1)
    # Start training.
    for step in range(FLAGS.max_steps):
        batch = mnist.train.next_batch(50)
train_step.run(feed_dict={x: batch[0], y_: batch[1]})
# Print the verification precision rate every 10 steps.
        if step % 10 == 0:
            summary, acc = sess.run([merged, accuracy], feed_dict={x: mnist.test.images, y_:
mnist.test.labels})
            test_writer.add_summary(summary, step)
            print('training accuracy is:', acc)
    print('Done training!')
    # Save the model to the model directory of the given train_url.
    builder = tf.saved_model.builder.SavedModelBuilder(os.path.join(FLAGS.train_url, 'model'))
    # Save parameter information of the model.
    tensor_info_x = tf.saved_model.utils.build_tensor_info(x)
    tensor_info_y = tf.saved_model.utils.build_tensor_info(y)
    # Define the signature (providing input, output, and method information) as the input parameter
for saving the model.
    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))
    # Import the graph information and variables.
    # The first parameter is transferred to the current session, including the graph structure and all 
variables.
    # The second parameter is a label for the meta graph to be saved. The label name can be 
customized. Here, the system-defined parameter is used.
    # The third parameter is used to save the signature.
    # main_op performs the Op or Ops group operation when loading a graph. When main_op is 
specified, it will run after the Op is loaded and recovered.
    # Run the initialization operation.
    # If strip_default_attrs is True, the default value attribute is deleted from the definition node.
    builder.add_meta_graph_and_variables(
            sess, [tf.saved_model.tag_constants.SERVING],
            signature_def_map={
                    'predict_images':
                            prediction_signature,
            },
            main_op=tf.tables_initializer(),
            strip_default_attrs=True)
    # Save the model.
    builder.save()

    print('Done exporting!')
if __name__ == '__main__':
    tf.app.run(main=main)
```

Inference code overview: Inference code inherits the TfServingBaseService class of the inference service and provides the preprocess and postprocess methods. The preprocess method is used to preprocesse the inputted images. The preprocessed images are transferred to the network model for final output. The model output result is transferred to the **postprocess** function for postprocessing. The postprocessed result is the final output result on the GUI.

The following is inference code. The source code is stored in the following path: modelarts-datasets-and-source-code/custom-basic-algorithms-for-deep learning/native-TensorFlow-for-handwritten-digit-recognition/code/customize_service_mnist.py

```
from PIL import Image
import numpy as np
import tensorflow as tf
from model_service.tfserving_model_service import TfServingBaseService

class mnist_service(TfServingBaseService):
    # Read images and data information, preprocess the images, and resize each image to 1,784. Save 
image information to
    # preprocessed_data and return preprocessed_data.
    def _preprocess(self, data):
        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, 784))
                preprocessed_data[k] = image1
        return preprocessed_data
    # Postprocess the logits value returned by the model. The prediction result is the class label 
corresponding to the maximum logits value, that is,
    # the prediction label of the image. The format is {'predict label': label_name}.
    def _postprocess(self, data):

        outputs = {}
        logits = data['scores'][0]
        label = logits.index(max(logits))
        outputs['predict label'] = label
return outputs
```

The following is the configuration file. The source code is stored in the following path: modelarts-datasets-and-source-code/custom-basic-algorithms-for-deep learning/native-TensorFlow-for-handwritten-digit-recognition/code/config.json

The config.json file contains four mandatory fields: model_type, metrics, model_algorithm, and apis.

Model_type: AI engine of the model, indicating the computing framework used by the model.

Metrics: model precision

Model_algorithm: model algorithm, indicating the usage of the model.

Apis: API arrays provided by the model for external systems.

Dependencies (optional): dependency packages of inference code and the model.

The reference is as follows:

```
{
        "model_type":"TensorFlow",
          # Model precision information, including the F1 score, accuracy, precision, and recall. The 
information is not mandatory for training MNIST.
        "metrics":{
                "f1":0.61185,
                "accuracy":0.8361458991671805,
                "precision":0.4775016224869111,
                "recall":0.8513980485387226
        },
          # Dependency packages required for inference
        "dependencies":[
                {
                        "installer":"pip",
                        "packages":[
                                {
                                        "restraint":"ATLEAST",
                                        "package_version":"1.15.0",
                                        "package_name":"numpy"
                                },
                                {
                                        "restraint":"",
                                        "package_version":"",
                                        "package_name":"h5py"
                                },
                                {
```

```
                    "restraint":"ATLEAST",
                    "package_version":"1.8.0",
                    "package_name":"tensorflow"
                },
                {
                    "restraint":"ATLEAST",
                    "package_version":"5.2.0",
                    "package_name":"Pillow"
                }
            ]
        }
    ],
     # Type of the model algorithm. In this example, the image classification model is used.
    "model_algorithm":"image_classification",
    "apis":[
        {
            "procotol":"错误!超链接引用无效。",
            "url":"/",
            "request":{
                "Content-type":"multipart/form-data",
                "data":{
                    "type":"object",
                    "properties":{
                        "images":{
                            "type":"file"
                        }
                    }
                }
            },
            "method":"post",
            "response":{
                "Content-type":"multipart/form-data",
                "data":{
                    "required":[
                        "predicted_label",
                        "scores"
                    ],
                    "type":"object",
                    "properties":{
                        "predicted_label":{
                            "type":"string"
                        },
                        "scores":{
                            "items":{
                                "minItems":2,
                                "items":[
                                    {
                                        "type":"string"
                                    },
                                    {
                                        "type":"number"
                                    }
                                ],
                                "type":"array",
                                "maxItems":2
                            },
                            "type":"array"
                        }
                    }
                }
            }
        }
    ]
}
```

Step 2 Upload scripts.

Upload the training script to OBS.

For details about how to upload files to OBS, see https://support.huaweicloud.com/en-us/modelarts_faq/modelarts_05_0013.html.

In this example, the upload path is **/modelarts-demo/codes/**.

NOTE

The file path cannot contain Chinese characters.

### 4.4.1.3 Training a Model

Step 1 Create a training job.

For details about the model training process, see section 3.3.1.2 "Training a Model." Parameter settings are as follows:

**Data Source**: Select the **MNIST** dataset or select the OBS path where the dataset is located.

Algorithm Source: Select Frequently-used framework.

AI Engine: Select TensorFlow and TF-1.13.1-python2.7.

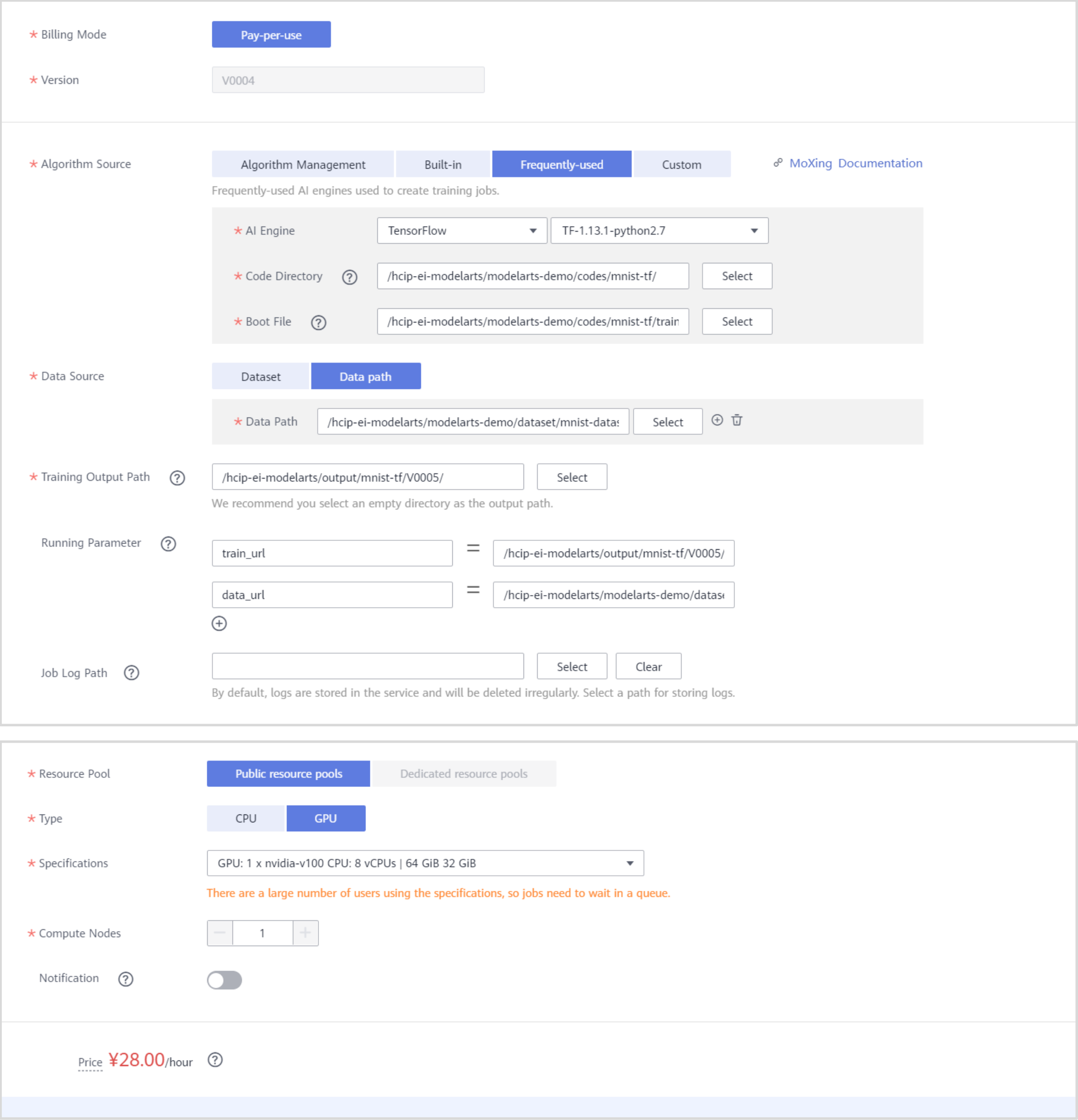
Code Directory: Select the parent path /modelarts-demo/codes/ of code.

**Boot File**: Select the boot script **train_mnist_tf.py**.

**Resource Pool**: This parameter is mandatory. Select a resource pool (including CPU and GPU) for the training job. GPU training is fast, and CPU training is slow. GPU/P100 is recommended.

**Compute Nodes**: Retain the default value **1**. (One node is used for standalone training, and more than one node is used for distributed training. Multi-node distributed training can accelerate the training process.)

Figure 4-5 shows the parameter settings. After setting the parameters, click **Next**. After confirming the parameter settings, click **Create Now**. The job is submitted.



* Billing Mode: Pay-per-use

* Version: V0004

* Algorithm Source: Algorithm Management | Built-in | Frequently-used | Custom — MoXing Documentation

Frequently-used AI engines used to create training jobs.

* AI Engine: TensorFlow | TF-1.13.1-python2.7

* Code Directory: /hcip-ei-modelarts/modelarts-demo/codes/mnist-tf/ Select

* Boot File: /hcip-ei-modelarts/modelarts-demo/codes/mnist-tf/train Select

* Data Source: Dataset | Data path

* Data Path: /hcip-ei-modelarts/modelarts-demo/dataset/mnist-datas Select

* Training Output Path: /hcip-ei-modelarts/output/mnist-tf/V0005/ Select

We recommend you select an empty directory as the output path.

Running Parameter:

train_url = /hcip-ei-modelarts/output/mnist-tf/V0005/

data_url = /hcip-ei-modelarts/modelarts-demo/datase

Job Log Path: Select Clear

By default, logs are stored in the service and will be deleted irregularly. Select a path for storing logs.

* Resource Pool: Public resource pools | Dedicated resource pools

* Type: CPU | GPU

* Specifications: GPU: 1 x nvidia-v100 CPU: 8 vCPUs | 64 GiB 32 GiB

There are a large number of users using the specifications, so jobs need to wait in a queue.

* Compute Nodes: 1

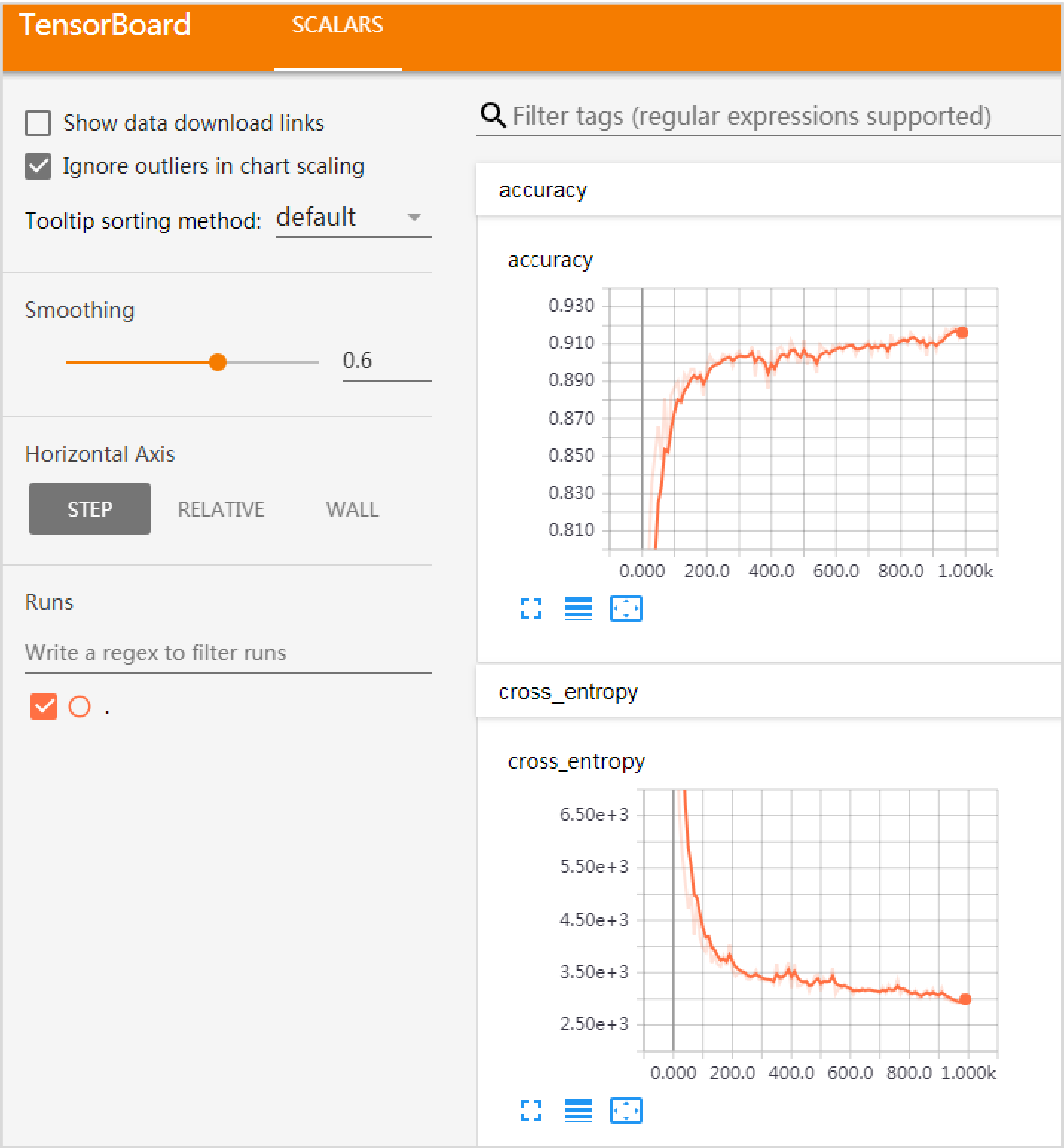
Notification

Price ¥28.00/hour

Figure 4-5 Parameter settings of the training job

Step 2 Create a visualization job.

For details, see Create a visualization job. in section 3.3.1.2 "Training a Model." The following figure shows the visualization job page.



☐ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

☑ ○ .

**Figure 4-6 Visualization job page**

Step 3 Upload scripts.

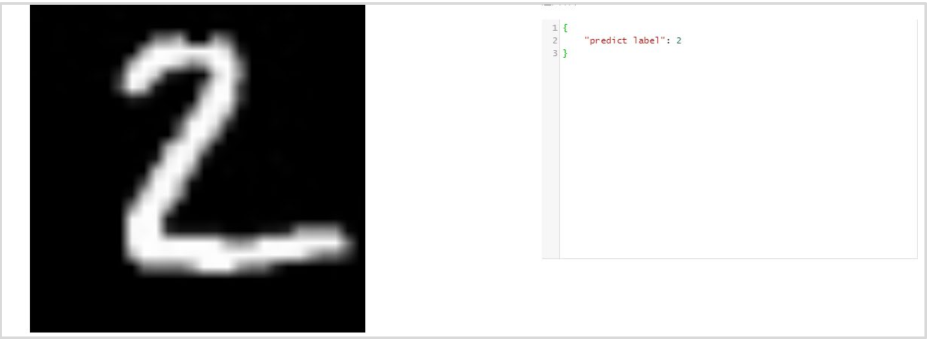
After the training job is complete, rename **customize_service_mnist.py** to **customize_service.py**, and upload the **customize_service.py** and **config.json** files to the **model** directory in the training output path (OBS path specified during training job creation) for model deployment.

### 4.4.1.4 Managing Models

For details, see section 3.3.1.3 "Managing a Model."

### 4.4.1.5 Deploying a Model

For details, see section 3.3.1.4 "Deploying a Model." The standard image format for image prediction is a gray handwritten digit image (28 x 28 pixels). If images do not meet format requirements, the prediction result may be inaccurate. Figure 4-7 shows the test result of the image in the following path: **modelarts-datasets-and-source-code/custom-basic-algorithms-for-deeplearning/native-TensorFlow-for-handwritten-digit-recognition/code/test-data/2.PNG**



```
{
    "predict label": 2
}
```

**Figure 4-7 Image prediction**

## 4.4.2 Using MoXing-TensorFlow for Flower Recognition

This section describes how to use MoXing custom scripts to perform distributed model training, deployment, and prediction on ModelArts. This section uses MoXing as an example to describe how to training flowers data. The procedure consists of five parts:

**Preparing data**: Create and Label the flowers dataset.

**Compiling scripts**: Use the MoXing framework to compile model training scripts.

**Training a model**: Use the compiled script to train the flowers dataset to obtain a well-trained model.

**Managing a model**: Import the model for deployment.

**Deploying a model**: Deploy a model as a real-time service.

### 4.4.2.1 Preparing Data

The data has been prepared in section 2.3.1 "Data Labeling for Flower Recognition".

### 4.4.2.2 Compiling Scripts

Scripts include training script **flowers_mox.py**, inference script **customize_service_flowers.py**, and configuration file **config.json**. The inference script and the configuration file will be used during model deployment. The configuration file is automatically generated during training. You need to upload the inference script.

Step 1 Interpret code.

Training code overview: Training code uses MoXing to train the flowers dataset. Both distributed training and standalone training are supported. The dataset has 50 images of five types. The **resnet_v1_50** model is used to classify the images into five types.

The following is training code. The source code is stored in the following path: modelarts-datasets-and-source-code/custom-basic-algorithms-for-deep learning/MoXing-TensorFlow-for-flower-recognition/code/flowers_mox.py

Training code is as follows:

```
# coding:utf-8
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
# Import the package required for training.
import os
import math
import numpy as np
import h5py
import tensorflow as tf
import moxing.tensorflow as mox
from moxing.tensorflow.optimizer import learning_rate_scheduler
from moxing.tensorflow.builtin_algorithms.metrics import write_config_json
from moxing.framework.common.data_utils.read_image_to_list import get_image_list
from moxing.framework.common.metrics.object_detection_metrics import get_metrics
from moxing.tensorflow.datasets.raw.raw_dataset import ImageClassificationRawFilelistDataset
from moxing.tensorflow.datasets.raw.raw_dataset import ImageClassificationRawFilelistMetadata
from moxing.tensorflow.builtin_algorithms.multilabels_metrics import process_with_class_metric
from moxing.tensorflow.builtin_algorithms.multilabels_metrics import post_process_fn_with_metric
# Define a dataset path.
tf.flags.DEFINE_string('data_url', default=None, help='dataset directory')
# Define the batch size of images to be trained, that is, the number of images trained in each step.
tf.flags.DEFINE_integer('batch_size', default=32, help='batch size per device per worker')
# Define the number of GPUs used for training. The default value is 1.
tf.flags.DEFINE_integer('num_gpus', default=1, help='number of gpus for training')
# Define a running mode. The default value is the training mode.
tf.flags.DEFINE_string('run_mode', default=mox.ModeKeys.TRAIN, help='Optional. run_mode. Default
to TRAIN')
# Define a model save path.
tf.flags.DEFINE_string('train_url', default=None, help='train dir')
# Define a training model name. The default value is resnet_v1_50.
tf.flags.DEFINE_string('model_name', default='resnet_v1_50', help='model_name')
# Define an image size during model training. The value of resnet_v1_50 is 224.
tf.flags.DEFINE_integer('image_size', default=None, help='Optional. Resnet_v1 use `224`.')
# Define the optimizer used for model training.
tf.flags.DEFINE_string('optimizer', default='sgd', help='adam or momentum or sgd, if None, sgd will
be used.')
# Define momentum.
tf.flags.DEFINE_float('momentum', default=0.9, help='Set 1 to use `SGD` opt, <1 to use momentum
opt')
# Define a dataset split ratio. The default ratio of splitting a dataset into a training set and a
validation set is 0.8:0.2.
tf.flags.DEFINE_string('split_spec', default='train:0.8,eval:0.2',
                       help='dataset split ratio. Format: train:0.8,eval:0.2')
# Define a learning rate. By default, the learning rate is 0.01 for the first 800 epochs, and is 0.001 for
800 to 1000 epochs.
tf.flags.DEFINE_string('learning_rate_strategy', default='800:0.01,1000:0.001',
                       help='Necessary. Learning rate decay strategy. Fotmat: 10:0.001,20:0.0001'
                       ' which means from epoch 0~10 use learning rate = 0.01 and from epoch
10~20 ')
flags = tf.flags.FLAGS

def main(*args):
  # Container cache path, which is used to store models
  cache_train_dir = '/cache/train_url'
  # If the path does not exist, create a path.
  if not mox.file.exists(cache_train_dir):
    mox.file.make_dirs(cache_train_dir)
  # Obtain the number of training nodes.
  num_workers = len(mox.get_flag('worker_hosts').split(','))
  # Obtain the number of GPUs.
  num_gpus = mox.get_flag('num_gpus')
  # Set the parameter update mode to parameter_server.
  mox.set_flag('variable_update', 'parameter_server')
  # Obtain meta information about the model.
  model_meta = mox.get_model_meta(flags.model_name)
  # Obtain a list of datasets.
  data_list, _, _ = get_image_list(data_path=flags.data_url, split_spec=1)
  # Define an image size during training.
  image_size = [flags.image_size, flags.image_size] if flags.image_size is not None else None
  # Define a data enhancement method.
  # mode: training or validation. The data enhancement methods vary depending on the mode.
  # model_name: model name
  # output_height: output image height. The default value is 224 for resnet_v1_50.
  # output_width: output image width. The default value is 224 for resnet_v1_50.
  def augmentation_fn(mode):
data_augmentation_fn = mox.get_data_augmentation_fn(
name=flags.model_name,
    run_mode=mode,
    output_height=flags.image_size or model_meta.default_image_size,
    output_width=flags.image_size or model_meta.default_image_size)
    return data_augmentation_fn

  # Obtain metadata information about the dataset.
  # data_list: list of datasets
  # split_spec: split ratio of the training set and validation set
  train_dataset_meta = eval_dataset_meta =
ImageClassificationRawFilelistMetadata(data_list=data_list,
split_spec=flags.split_spec)
  # Create a training set and a validation set.
  # metadata: metadata of the stored dataset
  # batch_size: number of images read each time
  # image_size: image size during model training. The default value is 224*224 for resnet_v1_50.
  # augmentation_fn: image enhancement function
  # num_readers: number of threads for reading data
  # preprocess_threads: number of threads for data processing
  # shuffle: whether to shuffle data
  # drop_remainder: whether to skip the batch when the number of images is insufficient in the last
batch
  train_dataset =ImageClassificationRawFilelistDataset(
  metadata=train_dataset_meta,
  batch_size=flags.batch_size    * mox.get_flag('num_gpus'),
  image_size=image_size,
augmentation_fn=augmentation_fn(mox.ModeKeys.TRAIN),
  drop_remainder=True)

  eval_dataset = ImageClassificationRawFilelistDataset(
  metadata=eval_dataset_meta,
  mode=mox.ModeKeys.EVAL,
  batch_size=flags.batch_size * mox.get_flag('num_gpus'),
  num_readers=1,
  shuffle=False,
  image_size=image_size,
  preprocess_threads=1,
  reader_kwargs={'num_readers': 1, 'shuffle': False},
augmentation_fn=augmentation_fn(mox.ModeKeys.EVAL),
  drop_remainder=True)
  # Read the number of images in the training set and the validation set.
  num_train_samples = train_dataset.total_num_samples
  num_eval_samples = eval_dataset.total_num_samples
  num_classes = train_dataset_meta.num_classes
  labels_dict = train_dataset_meta.labels_dict
  label_map_dict = train_dataset_meta.label_map_dict
  # Write the index file. This file is used to save information required for model inference. The
information saved here is a label name list, which is used for storing
# the real label category outputted during inference prediction. (The label used in the training is one-
hot encoded information, and the real label is not saved.)
  index_file = h5py.File(os.path.join(cache_train_dir, 'index'), 'w')
  index_file.create_dataset('labels_list', data=[np.string_(i) for i in
train_dataset_meta.labels_dict.keys()])
  index_file.close()
  # batch_size quantity on each machine.
  batch_size_per_device = flags.batch_size or int(round(math.ceil(min(
num_train_samples / 10.0 / num_gpus / num_workers, 16))))
  # Total batch_size.
  total_batch_size = batch_size_per_device * num_gpus * num_workers
  # Total number of training epochs.
  max_epochs = float(flags.learning_rate_strategy.split(',')[-1].split(':')[0])
  # Number of training steps.
  max_number_of_steps = int(round(math.ceil(
    max_epochs * num_train_samples / float(total_batch_size))))
  tf.logging.info('Total steps = %s' % max_number_of_steps)


  # Define a data read function.
  def input_fn(run_mode, **kwargs):
    if run_mode == mox.ModeKeys.EVAL:
        dataset = eval_dataset
    elif run_mode == mox.ModeKeys.TRAIN:
        dataset = train_dataset
    else:
        raise ValueError('Unsupported run mode. Only `TRAIN` and `EVAL` are supported. ')

    image_name, image, label = dataset.get(['image_name', 'image', 'label'])
    return mox.InputSpec(split_to_device=True).new_input(inputs=[image_name, image, label])


# Define postprocessing operations for validation, calculate metrics of the validation set, such as
recall, precision, accuracy, and mean_ap, and write them into the metric.json and config.json files.
  def multiclass_post_process_fn_with_metric(outputs):
    output_metrics_dict = post_process_fn_with_metric(outputs)
    post_metrics_dict = process_with_class_metric(labels_dict, output_metrics_dict, label_map_dict)
    get_metrics(cache_train_dir, post_metrics_dict)
    write_config_json(metrics_dict=post_metrics_dict['total'],
                                        train_url= cache_train_dir,
                                        model_algorithm='image_classification',
                                        inference_url= cache_train_dir)
    results = {'accuracy': post_metrics_dict['total']['accuracy']}

    return results


  # Implement the model and return a ModelSpec instance.
  def model_fn(inputs, run_mode, **kwargs):
    image_names, images, labels = inputs

    if run_mode == mox.ModeKeys.EXPORT:
      images = tf.placeholder(dtype=images.dtype, shape=[None, None, None, 3],
name='images_ph')
    image_size = flags.image_size or model_meta.default_image_size

    mox_model_fn = mox.get_model_fn(
      name=flags.model_name,
      run_mode=run_mode,
      num_classes=num_classes,
      batch_norm_fused=True,
      batch_renorm=False,
      image_height=image_size,
      image_width=image_size)
    # Model output value.
    logits, end_points = mox_model_fn(images)
    # Process the label value. The 1/k processing is performed for k-hot label, which is obtained from
the related paper.
    labels_one_hot = tf.divide(labels, tf.reduce_sum(labels, 1, keepdims=True))
    # Calculate a cross-entropy loss.
    loss = tf.losses.softmax_cross_entropy(labels_one_hot, logits=logits, label_smoothing=0.0,
weights=1.0)
    # Calculate a regularization loss.
    regularization_losses = mox.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
    if len(regularization_losses) > 0:
      regularization_loss = tf.add_n(regularization_losses)
      loss = loss + regularization_loss
    log_info = {'loss': loss}

    inputs_dict = {'images': images}
    outputs_dict = {'logits': logits}

    export_spec = mox.ExportSpec(inputs_dict=inputs_dict,
                                 outputs_dict=outputs_dict,
                                 version='model')
    # LogEvaluationMetricHook monitoring information
    monitor_info = {'loss': loss, 'logits': logits, 'labels': labels, 'image_names': image_names}

    # LogEvaluationMetricHook is used to verify the validation set during training and view the
model training effect.
    # monitor_info: records and summarizes information.
    # batch_size: used to calculate epochs based on steps
    # samples_in_train: number of samples in the training set of each epoch
    # samples_in_eval: number of samples in the validation set of each epoch
    # num_gpus: number of GPUs. If the value is None, value 1 will be used by default.
    # num_workers: number of workers. If the value is None, value 1 will be used by default.
    # evaluate_every_n_epochs: Perform verification after n epochs are trained.
    # mode: Possible values are {auto, min, max}. In min mode, the training ends when the
monitoring metrics stop decreasing. In max mode, the training ends when the monitoring metrics
stop increasing. In auto mode, the system automatically infers the value from the name of the
monitoring metric.
    # prefix: prefix of the message whose monitor_info is to be printed
    # log_dir: directory for storing summary of monitor_info
    # device_aggregation_method: function for aggregating monitor_info between GPUs
    # steps_aggregation_method: function for aggregating monitor_info among different steps
    # worker_aggregation_method: function for aggregating monitor_info among different workers
    # post_process_fn: postprocesses monitor_info information.
    hook = mox.LogEvaluationMetricHook(
      monitor_info=monitor_info,
      batch_size=batch_size_per_device,
      samples_in_train=num_train_samples,
      samples_in_eval=num_eval_samples,
      num_gpus=num_gpus,
      num_workers=num_workers,
      evaluate_every_n_epochs=10,
      prefix='[Validation Metric]',
      log_dir=cache_train_dir,
      device_aggregation_method=mox.HooksAggregationKeys.USE_GPUS_ALL,
      steps_aggregation_method=mox.HooksAggregationKeys.USE_STEPS_ALL,
      worker_aggregation_method=mox.HooksAggregationKeys.USE_WORKERS_ALL,
      post_process_fn=multiclass_post_process_fn_with_metric)

    model_spec = mox.ModelSpec(loss=loss,
                               log_info=log_info,
                               output_info=outputs_dict,
                               export_spec=export_spec,
                               hooks=hook)
    return model_spec
  # Define an optimization function.
  def optimizer_fn():
    global_batch_size = total_batch_size * num_workers
    lr = learning_rate_scheduler.piecewise_lr(flags.learning_rate_strategy,
                                                      num_samples=num_train_samples,
                                                    global_batch_size=global_batch_size)
    # SGD optimization function
    if flags.optimizer is None or flags.optimizer == 'sgd':
      opt = mox.get_optimizer_fn('sgd', learning_rate=lr)()
    # Momentum optimization function
    elif flags.optimizer == 'momentum':
      opt = mox.get_optimizer_fn('momentum', learning_rate=lr, momentum=flags.momentum)()
    # Adam optimization function
    elif flags.optimizer == 'adam':
      opt = mox.get_optimizer_fn('adam', learning_rate=lr)()
    else:
      raise ValueError('Unsupported optimizer name: %s' % flags.optimizer)
    return opt

  mox.run(input_fn=input_fn,
          model_fn=model_fn,
          optimizer_fn=optimizer_fn,
          run_mode=flags.run_mode,
          inter_mode=mox.ModeKeys.EVAL,
                    batch_size=flags.batch_size,
                    log_dir= cache_train_dir,
                    auto_batch=False,
                    save_summary_steps=5,
                    max_number_of_steps= max_number_of_steps,
                    output_every_n_steps= max_number_of_steps,
                    export_model=mox.ExportKeys.TF_SERVING)
# The accuracy metrics of the validation set are written into the config.json file. After the training is complete, the file is copied to the model directory for model management.
    mox.file.copy_parallel(cache_train_dir, flags.train_url)
    mox.file.copy(os.path.join(cache_train_dir, 'config.json'),
                                os.path.join(flags.train_url, 'model', 'config.json'))
    mox.file.copy(os.path.join(cache_train_dir, 'index'),
                                os.path.join(flags.train_url, 'model', 'index'))

if __name__ == '__main__':
    tf.app.run(main=main)
```

Inference code overview: Inference code inherits the TfServingBaseService class of the inference service and provides the preprocess and postprocess methods. The preprocess method is used to preprocesse the inputted images. The preprocessed images are transferred to the network model for final output. The model output result is transferred to the **postprocess** function for postprocessing. The postprocessed result is the final output result on the GUI.

The following is inference code. The source code is stored in the following path: modelarts-datasets-and-source-code/custom-basic-algorithms-for-deep learning/MoXing-TensorFlow-for-flower-recognition/code/customize_service_flowers.py

```
from PIL import Image
import h5py
import numpy as np
import os
from model_service.tfserving_model_service import TfServingBaseService

class cnn_service(TfServingBaseService):
    # Read images and data information and preprocess the images.
    def _preprocess(self, data):
        preprocessed_data = {}
        for k, v in data.items():
            for file_name, file_content in v.items():
                image = Image.open(file_content)
                image = image.convert('RGB')
                image = np.asarray(image, dtype=np.float32)
                image = image[np.newaxis, :, :, :]
                preprocessed_data[k] = image
return preprocessed_data

    # Postprocess the return value of the model and return the prediction result.
    def _postprocess(self, data):
        h5f = h5py.File(os.path.join(self.model_path, 'index'), 'r')
        labels_list = h5f['labels_list'][:]
        h5f.close()
        outputs = {}
    # Define the softmax function.
    def softmax(x):
      x = np.array(x)
      orig_shape = x.shape

      if len(x.shape) > 1:
        # Matrix
        exp_minmax = lambda x: np.exp(x - np.max(x))
        denom = lambda x: 1.0 / np.sum(x)
        x = np.apply_along_axis(exp_minmax, 1, x)
        denominator = np.apply_along_axis(denom, 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

    # Perform softmax processing on the return value of the model.
    predictions_list = softmax(data['logits'][0])
    predictions_list = ['%.3f' % p for p in predictions_list]
    # Sort the results.
    scores = dict(zip(labels_list, predictions_list))
scores = sorted(scores.items(), key=lambda item: item[1], reverse=True)
# Return the category labels with top 5 reliability.
    if len(labels_list) > 5:
      scores = scores[:5]
    label_index = predictions_list.index(max(predictions_list))
    predicted_label = str(labels_list[label_index])
    print('predicted label is: %s ' % predicted_label)
    outputs['predicted_label'] = predicted_label
    outputs['scores'] = scores
    return outputs
```

For details about the configuration file, see section 4.4.1.2 "Compiling Scripts." The values of four precision-related metrics are automatically generated during the training.

Step 2 Upload scripts.

Upload the training script to OBS. In this example, the upload path is **/modelarts-demo/codes/**.

NOTE

The file path cannot contain Chinese characters.

For details about how to upload data, see https://support.huaweicloud.com/en-us/modelarts_faq/modelarts_05_0013.html.

### 4.4.2.3 Training a Model

Step 1 Create a training job.

For details about the model training process, see section 3.3.1.2 "Training a Model." Parameter settings are as follows:

**Data Source**: Select the flower recognition dataset generated in section **Data Management**.

Algorithm Source: Select Frequently-used.

AI Engine: Select TensorFlow and TF-1.8.0-python2.7.

Code Directory: Select the parent path /modelarts-demo/codes/ of code.

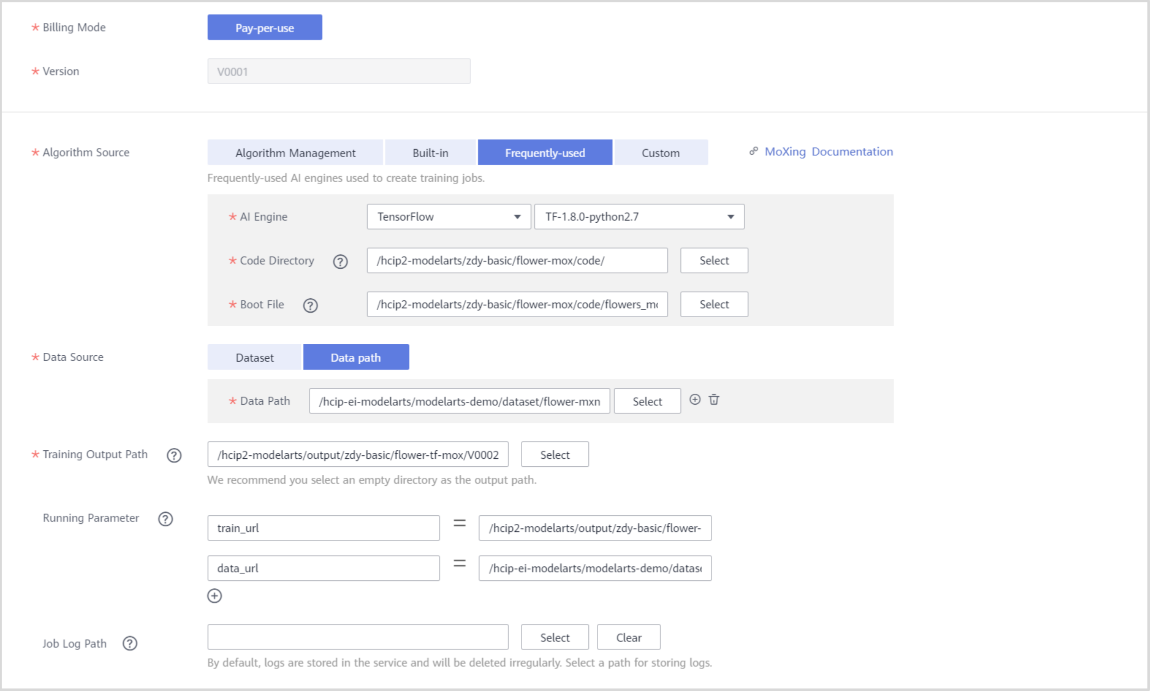
**Boot File**: Select the boot script **flowers_mox.py**.

**Resource Pool**: Select a resource pool (including CPU and GPU) for the training job. GPU training is fast, and CPU training is slow. GPU/P100 is recommended.

Training Output Path: /modelarts-demo/output/flowers_mox/

**Compute Nodes**: Set it to **2**. (One node is used for standalone training, and more than one node is used for distributed training. Multi-node distributed training can accelerate the training process.)

The following figure shows the parameter settings. After setting the parameters, click **Next**. After confirming the parameter settings, click **Create Now**. The job is submitted.



* Billing Mode: Pay-per-use

* Version: V0001

* Algorithm Source: Frequently-used

Frequently-used AI engines used to create training jobs.

* AI Engine: TensorFlow TF-1.8.0-python2.7

* Code Directory: /hcip2-modelarts/zdy-basic/flower-mox/code/

* Boot File: /hcip2-modelarts/zdy-basic/flower-mox/code/flowers_m

* Data Source: Data path

* Data Path: /hcip-ei-modelarts/modelarts-demo/dataset/flower-mxn

* Training Output Path: /hcip2-modelarts/output/zdy-basic/flower-tf-mox/V0002

We recommend you select an empty directory as the output path.

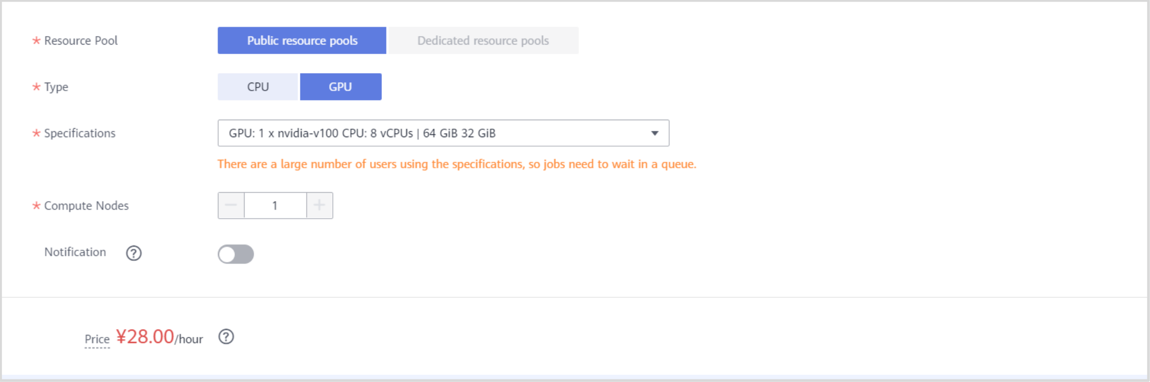
Running Parameter:

train_url = /hcip2-modelarts/output/zdy-basic/flower-

data_url = /hcip-ei-modelarts/modelarts-demo/datas

Job Log Path:

By default, logs are stored in the service and will be deleted irregularly. Select a path for storing logs.



* Resource Pool: Public resource pools | Dedicated resource pools

* Type: CPU | GPU

* Specifications: GPU: 1 x nvidia-v100 CPU: 8 vCPUs | 64 GiB 32 GiB

There are a large number of users using the specifications, so jobs need to wait in a queue.

* Compute Nodes: 1

Notification

Price ¥28.00/hour

**Figure 4-8 Parameter settings of the training job**

Step 2 Create a visualization job.

For details, see Create a visualization job. 4 in section 3.3.1.2 "Training a Model."

Step 3 Upload scripts.

After the training job is complete, rename **customize_service_flowers.py** to **customize_service.py** and upload it to the **model** directory in the training output path (OBS path specified during training job creation).

### 4.4.2.4 Managing Models

For details, see section 3.3.1.3 "Managing a Model."

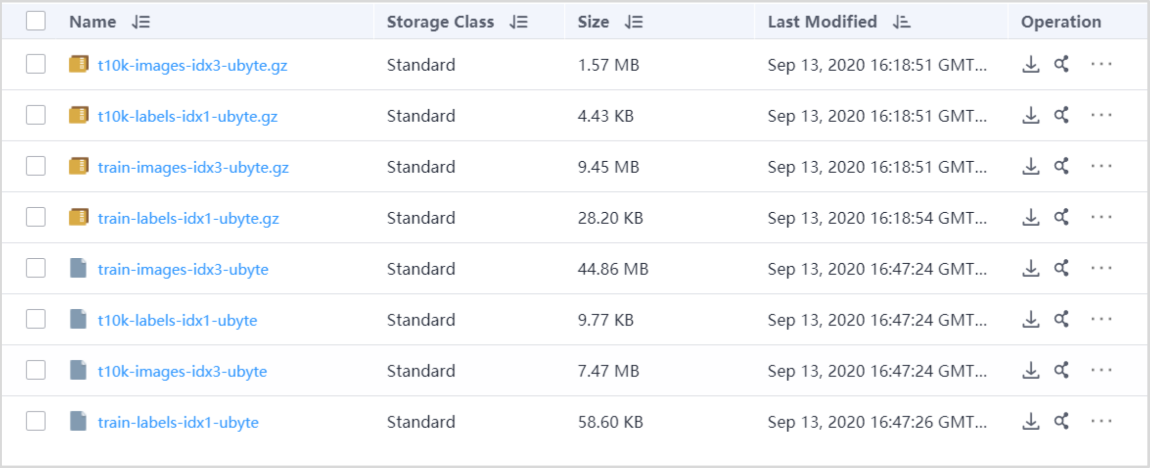
### 4.4.2.5 Deploying a Model

For details, see section 3.3.1.4 "Deploying a Model."

## 4.4.3 Using Native MXNet for Handwritten Digit Recognition

This experiment describes how to use MXNet to implement handwritten digit recognition, deploy and test a model, and use visualization jobs in the training process.

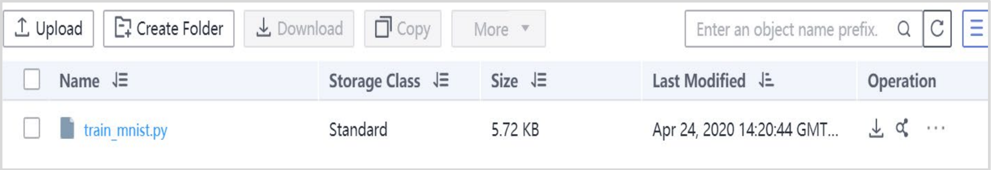
Step 1 Upload the MNIST dataset to the OBS bucket using the method described in section 2.3.3. See the following figure.



| □ | Name | Storage Class | Size | Last Modified | Operation |
|---|---|---|---|---|---|
| □ | t10k-images-idx3-ubyte.gz | Standard | 1.57 MB | Sep 13, 2020 16:18:51 GMT... | ··· |
| □ | t10k-labels-idx1-ubyte.gz | Standard | 4.43 KB | Sep 13, 2020 16:18:51 GMT... | ··· |
| □ | train-images-idx3-ubyte.gz | Standard | 9.45 MB | Sep 13, 2020 16:18:51 GMT... | ··· |
| □ | train-labels-idx1-ubyte.gz | Standard | 28.20 KB | Sep 13, 2020 16:18:54 GMT... | ··· |
| □ | train-images-idx3-ubyte | Standard | 44.86 MB | Sep 13, 2020 16:47:24 GMT... | ··· |
| □ | t10k-labels-idx1-ubyte | Standard | 9.77 KB | Sep 13, 2020 16:47:24 GMT... | ··· |
| □ | t10k-images-idx3-ubyte | Standard | 7.47 MB | Sep 13, 2020 16:47:24 GMT... | ··· |
| □ | train-labels-idx1-ubyte | Standard | 58.60 KB | Sep 13, 2020 16:47:26 GMT... | ··· |

**Figure 4-9 MNIST file**

Step 2 Upload the code file **train_mnist.py** to the OBS bucket. For example, upload **train_mnist.py** to the **modelarts-demo/builtin-algorithm/mxnet_mxnet** folder in the OBS path, as shown in the following figure. The source code of **train_mnist.py** is stored in the following path: **modelarts-datasets-and-source-code/custom-basic-algorithms-for-deep learning/native-MXNet-for-handwritten-digit-recognition/code/train_mnist.py**



| □ | Name | Storage Class | Size | Last Modified | Operation |
|---|---|---|---|---|---|
| □ | train_mnist.py | Standard | 5.72 KB | Apr 24, 2020 14:20:44 GMT... | ⤓ ⋯ |

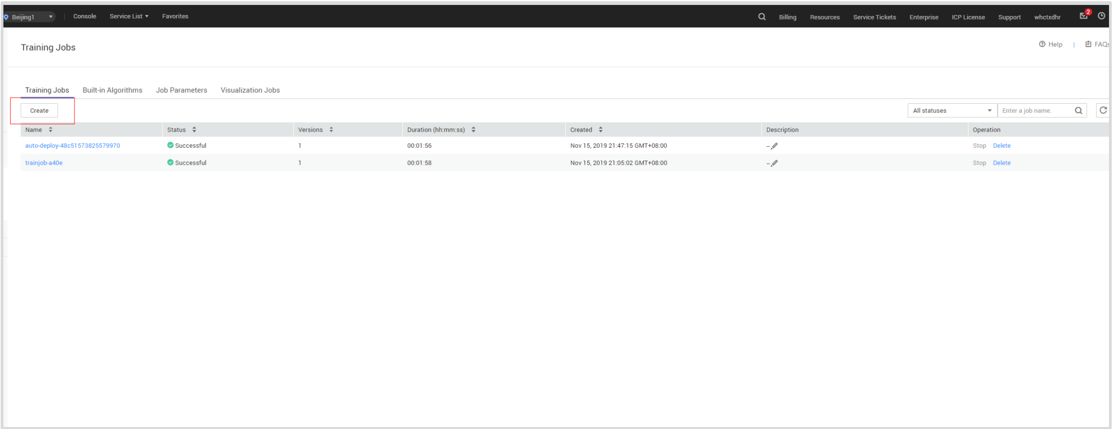
**Figure 4-10 Uploading code to OBS**

The code of the training script **train_mnist.py** is interpreted as follows:

```
# The script uses the native MXNet framework to train the MNIST dataset, which contains 60,000
# white and black images (28 x 28 pixels), with accuracy of about 99% in the training set.
import mxnet as mx
import argparse
import logging
import os

# Define input parameters.
parser = argparse.ArgumentParser(description="train mnist",
                                 formatter_class=argparse.ArgumentDefaultsHelpFormatter)
# Number of classes. In this example, handwritten digits are used, so the value is 10.
parser.add_argument('--num_classes', type=int, default=10,
                        help='the number of classes')
# Number of samples, which is used for lr change. The MNIST training set contains 60,000 images.
parser.add_argument('--num_examples', type=int, default=60000,
                        help='the number of training examples')

# data_url indicates the data storage path of the data source on the GUI. It is a path of s3://.
parser.add_argument('--data_url', type=str, default=None,
                         help='the training data')
# Learning rate, which is the step of parameter update each time
parser.add_argument('--lr', type=float, default=0.05,
                        help='initial learning rate')
# Epochs to be trained. When all datasets enter the model once, it is called an epoch.
parser.add_argument('--num_epochs', type=int, default=10,
                        help='max num of epochs')
# Interval for outputting batch logs.
parser.add_argument('--disp_batches', type=int, default=20,
                        help='show progress for every n batches')
# Parameters of a model are updated each time batch_size of data is processed. This is called a
batch.
parser.add_argument('--batch_size', type=int, default=128,
                        help='the batch size')
parser.add_argument('--kv_store', type=str, default='device',
                        help='key-value store type')
# File output path, that is, the training output path displayed on the GUI. It is also a path of s3://.
```

```
parser.add_argument('--train_url', type=str, default=None,
                        help='the path model saved')
# Number of GPUs. The job delivers this parameter based on the machine specifications in the 
selected resource pool. If you use your own code, you only need to
# add this parameter to define the context.
parser.add_argument('--num_gpus', type=int, default='0',
                        help='number of gpus')
# Determine whether the generated code must be in a format that can be deployed as an inference 
service.
parser.add_argument('--export_model', type=int, default=1, help='1: export model for predict job \
                                                                      0: not export model')
args, unkown = parser.parse_known_args()

# Read data by using the MNISTIter API provided by MXNet. Because the dataset name in the market
# is train-images-idx3-ubyte, the path is Data storage location + Training file name.
def get_mnist_iter(args):
    train_image = os.path.join(args.data_url, 'train-images-idx3-ubyte')
    train_label = os.path.join(args.data_url, 'train-labels-idx1-ubyte')

    train = mx.io.MNISTIter(image=train_image,
                                  label=train_label,
                                  data_shape=(1, 28, 28),
                                  batch_size=args.batch_size,
                                  shuffle=True,
                                  seed=10)
    return train

# Construct a simple fully-connected network with activation functions.
def get_symbol(num_classes=10, **kwargs):
    # Initialize variables, which must be defined at the beginning of all networks.
    data = mx.symbol.Variable('data')
   # Flatten the input of [m, n] to [1, m*n].
data = mx.sym.Flatten(data=data)
   # Fully-connected layer. num_hidden indicates the number of neurons.
fc1   = mx.symbol.FullyConnected(data = data, name='fc1', num_hidden=128)
   # Activation function layer, which is used to add the non-linearity of the model.
    act1 = mx.symbol.Activation(data = fc1, name='relu1', act_type="relu")
    fc2   = mx.symbol.FullyConnected(data = act1, name = 'fc2', num_hidden = 64)
act2 = mx.symbol.Activation(data = fc2, name='relu2', act_type="relu")
   # The value of num_hidden is 10, because the final output is the probability of 10 digits.
fc3   = mx.symbol.FullyConnected(data = act2, name='fc3', num_hidden=num_classes)
   # Normalize the output of the FC layer to 0 to 1. The total probability of 10 classes is 1.
    mlp   = mx.symbol.SoftmaxOutput(data = fc3, name = 'softmax')
    return mlp

def fit(args):
   # Indicates whether distributed or standalone program is used.
    kv = mx.kvstore.create(args.kv_store)
  # Define the logging level and format.
    head = '%(asctime)-15s Node[' + str(kv.rank) + '] %(message)s'
    logging.basicConfig(level=logging.DEBUG, format=head)
    logging.info('start with arguments %s', args)
    # Obtain training data.
    train = get_mnist_iter(args)
    # Define that the current model is stored after each epoch of the MXNet ends.
    checkpoint = mx.callback.do_checkpoint(args.train_url if kv.rank == 0 else "%s-%d" % (
        args.train_url, kv.rank))
    # Define a callback after each batch is complete, including running speed information and the
mxboard file generated in the training output path. They can be used for deploying a visualization
job.
    batch_end_callbacks = [mx.contrib.tensorboard.LogMetricsCallback(
            args.train_url), mx.callback.Speedometer(args.batch_size,
                                                     args.disp_batches)]
    # Obtain the simple fully-connected network mentioned above.
    network = get_symbol(num_classes=args.num_classes)
    # Define whether to run on the GPU or CPU. The num_gpus  parameter is transferred by the
machine specifications when the job is started. You can directly use the parameter.
  # Define context in this cyclic list mode.
    devs = mx.cpu() if args.num_gpus == 0 else [mx.gpu(int(i)) for i in range(args.num_gpus)]
    # Create a model.
    model = mx.mod.Module(context=devs, symbol=network)
    # Define initialization functions of the model.
    initializer = mx.init.Xavier(rnd_type='gaussian', factor_type="in", magnitude=2)
    # Create optimizer parameters. In this example, simple initial learningrate and weightdecay are
used.
    optimizer_params = {'learning_rate': args.lr, 'wd' : 0.0001}
    # Run
    model.fit(train,# Train data.
                begin_epoch=0,# This parameter is used for checkpoint recovery. If the checkpoint is
loaded, this parameter is used.
                num_epoch=args.num_epochs,# Number of epochs for training
                eval_data=None,# Validation dataset
                eval_metric=['accuracy'],# Validation metric. In this example, the value is acc.
                kvstore=kv,# kvstore is used to control the standalone or distributed system. The
standalone system is used by default.
                optimizer='sgd',# Parameter update method. In this example, random gradient
descent is used.
                optimizer_params=optimizer_params,# It is used to control the changes of
parameters, for example, lr.
                initializer=initializer,# Model initialization function
                arg_params=None,# Model parameter. If the value is not None, the value comes
from the existing model.
                aux_params=None,# Auxiliary model parameter. If the value is not None, the value
comes from the existing model.
                batch_end_callback=batch_end_callbacks,# Function invoked after each batch ends
                epoch_end_callback=checkpoint,# Parameter invoked after each epoch ends
                allow_missing=True# Model parameter missing is allowed. If a model parameter is
missing, the initialization function is used.

# Perform the following operations if you want to deploy the model as a real-time service on
HUAWEI CLOUD ModelArts.
    if args.export_model == 1 and args.train_url is not None and len(args.train_url):
        end_epoch = args.num_epochs
        save_path = args.train_url if kv.rank == 0 else "%s-%d" % (args.train_url, kv.rank)
        params_path = '%s-%04d.params' % (save_path, end_epoch)
        json_path = ('%s-symbol.json' % save_path)
        logging.info(params_path + 'used to predict')
        pred_params_path = os.path.join(args.train_url, 'model', 'pred_model-0000.params')
        pred_json_path = os.path.join(args.train_url, 'model', 'pred_model-symbol.json')
    # MoXing is a Huawei-developed framework of ModelArts. In this example, the file API of MoX is used to access OBS.
            import moxing.mxnet as mox
    # copy indicates the file copy operation, and remove indicates the file deletion operation. For details, see mox.framework api.
    # The required file structure is generated in train_url (training output path).
    # |--train_url
    #       |--model
    #              xxx-0000.params
    #              xxx-symbol.json
            mox.file.copy(params_path, pred_params_path)
            mox.file.copy(json_path, pred_json_path)
            for i in range(1, args.num_epochs + 1, 1):
                mox.file.remove('%s-%04d.params' % (save_path, i))
            mox.file.remove(json_path)

if __name__ == '__main__':
    fit(args)
```

Step 3 On the ModelArts console, choose **Training Jobs** and click **Create**.

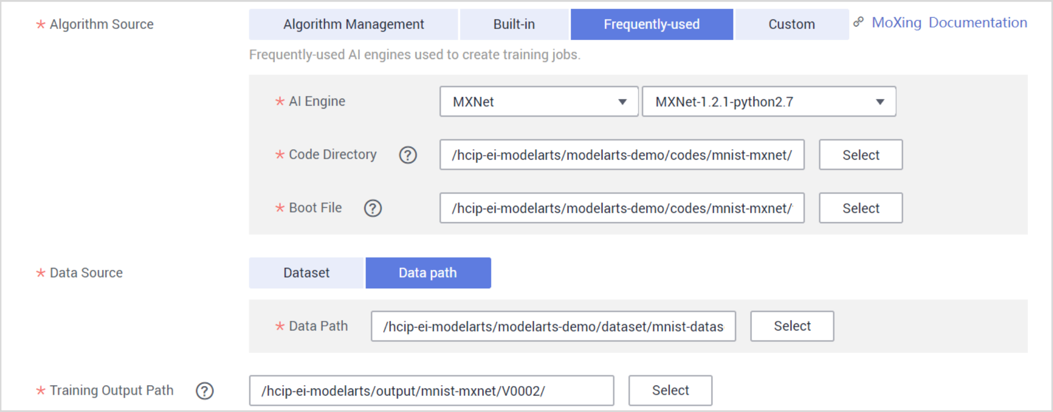


Training Jobs

| Name | Status | Versions | Duration (hh:mm:ss) | Created | Description | Operation |
|---|---|---|---|---|---|---|
| auto-deploy-48c51573825579970 | Successful | 1 | 00:01:56 | Nov 15, 2019 21:47:15 GMT+08:00 | -- | Stop Delete |
| trainjob-a40e | Successful | 1 | 00:01:58 | Nov 15, 2019 21:05:02 GMT+08:00 | -- | Stop Delete |

**Figure 4-11 Creating training jobs**

A job name must be unique. If the data source is a dataset imported from the market, select the corresponding dataset (you can view the dataset on the **Datasets** tab page of the **Data Management** page) or select the data storage location. In this example, the data is stored in the OBS path **modelarts-demo/data**. Select this path, as shown in the following figure.



* Algorithm Source: Algorithm Management | Built-in | **Frequently-used** | Custom — MoXing Documentation

Frequently-used AI engines used to create training jobs.

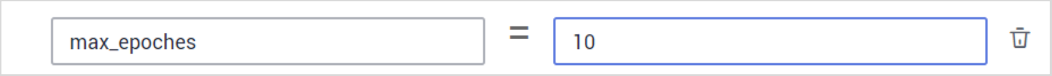
* AI Engine: MXNet | MXNet-1.2.1-python2.7
* Code Directory: /hcip-ei-modelarts/modelarts-demo/codes/mnist-mxnet/ [Select]
* Boot File: /hcip-ei-modelarts/modelarts-demo/codes/mnist-mxnet/ [Select]

* Data Source: Dataset | **Data path**
* Data Path: /hcip-ei-modelarts/modelarts-demo/dataset/mnist-datas [Select]

* Training Output Path: /hcip-ei-modelarts/output/mnist-mxnet/V0002/ [Select]

**Figure 4-12 Data selection**

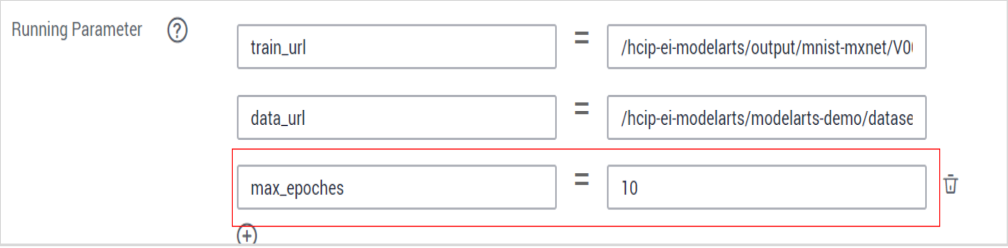
After selecting data, select **mxnet1.2.1-python2.7** in the frequently-used framework. Select the **modelarts-demo/builtin-algorithm/mxnet_mnist/** directory where code is stored, and select **train_mnist.py** as the boot file. Select an existing path to store the model output. Select **Public resource pools** for **Resource Pool** and click **Next**.



max_epoches = 10

**Figure 4-13 Parameter settings**

If any custom parameters need to be entered in code, you only need to define the corresponding **argparse** parsing in code, and enter the parameters in **Running Parameter**.



Running Parameter

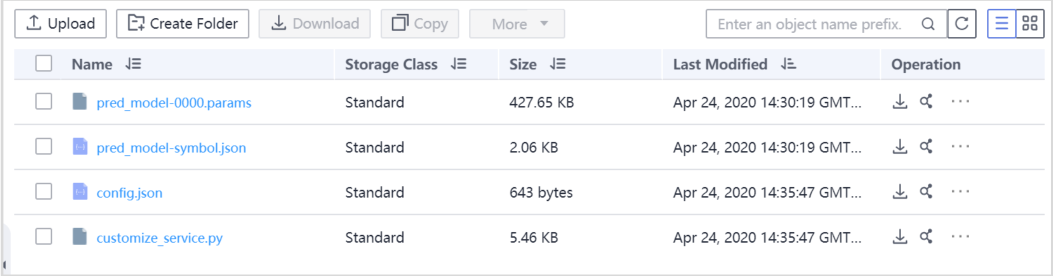
train_url = /hcip-ei-modelarts/output/mnist-mxnet/V0

data_url = /hcip-ei-modelarts/modelarts-demo/datase

max_epoches = 10

**Figure 4-14 Entering running parameters**

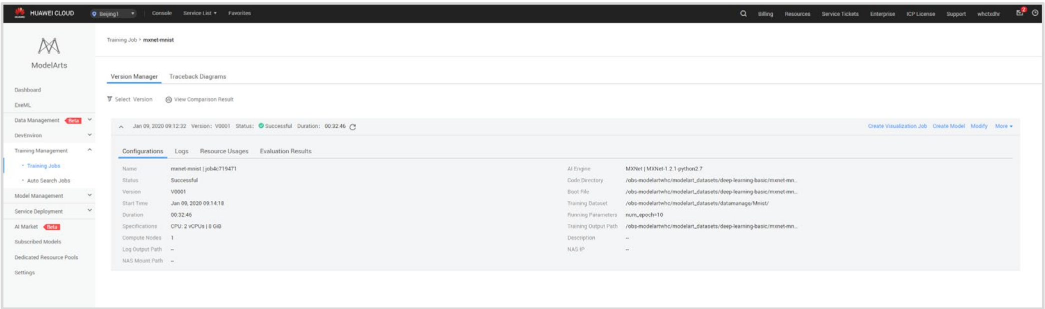
Step 4 After the training job is created, go to the corresponding job and wait until job running is complete. During the process, you can check logs and pay attention to the result. After the job is complete, you can view the result in **Training Output Path**. In this example, the selected OBS path is **modelarts-demo/result_log/mnist_mxnet_log**. The following figure shows the result.



| Name | Storage Class | Size | Last Modified | Operation |
|---|---|---|---|---|
| pred_model-0000.params | Standard | 427.65 KB | Apr 24, 2020 14:30:19 GMT... | |
| pred_model-symbol.json | Standard | 2.06 KB | Apr 24, 2020 14:30:19 GMT... | |
| config.json | Standard | 643 bytes | Apr 24, 2020 14:35:47 GMT... | |
| customize_service.py | Standard | 5.46 KB | Apr 24, 2020 14:35:47 GMT... | |

Figure 4-15 Training job output result

The **events** file is generated by the mxboard. The mxboard is a module provided by the MXNet to observe the accuracy and loss value changes during the training process. This file is used to deploy the visualization job. You can create a visualization job on the right of the training job to view the changes of parameters, for example, the precision loss of the model, as shown in the following figure.

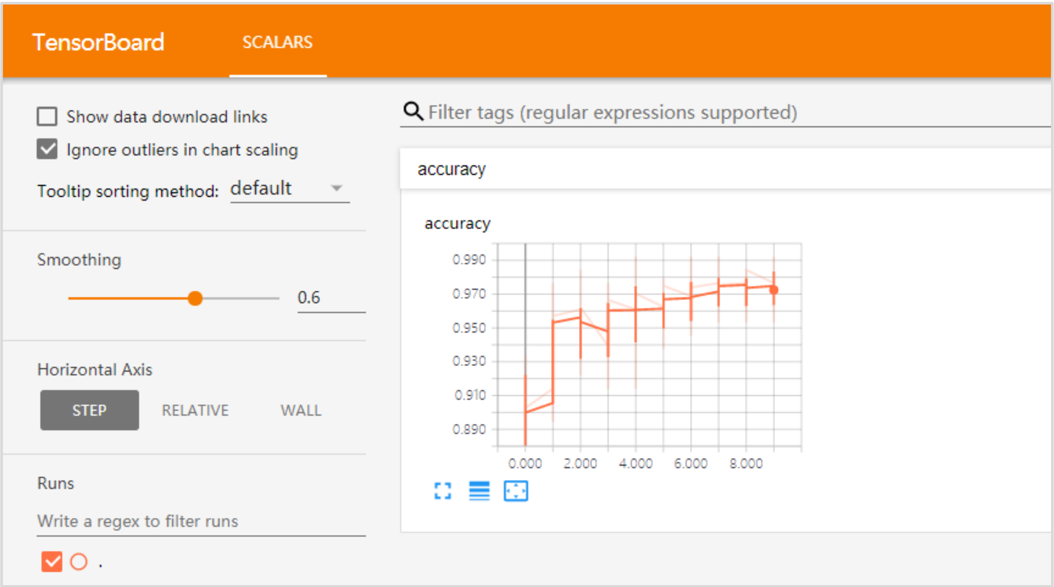


Jan 09, 2020 09:12:32 Version: V0001 Status: Successful Duration: 00:32:46

Name mxnet-mnist | job4c719471
Status Successful
Version V0001
Start Time Jan 09, 2020 09:14:18
Duration 00:32:46
Specifications CPU: 2 vCPUs | 8 GiB
Compute Nodes 1
Log Output Path --
NAS Mount Path --

AI Engine MXNet | MXNet-1.2.1-python2.7
Code Directory /obs-modelartwhc/modelart_datasets/deep-learning-basic/mxnet-mn...
Boot File /obs-modelartwhc/modelart_datasets/deep-learning-basic/mxnet-mn...
Training Dataset /obs-modelartwhc/modelart_datasets/datamanage/Mnist/
Running Parameters num_epoch=10
Training Output Path /obs-modelartwhc/modelart_datasets/deep-learning-basic/mxnet-mn...
Description --
NAS IP --

Figure 4-16 Creating a visualization job

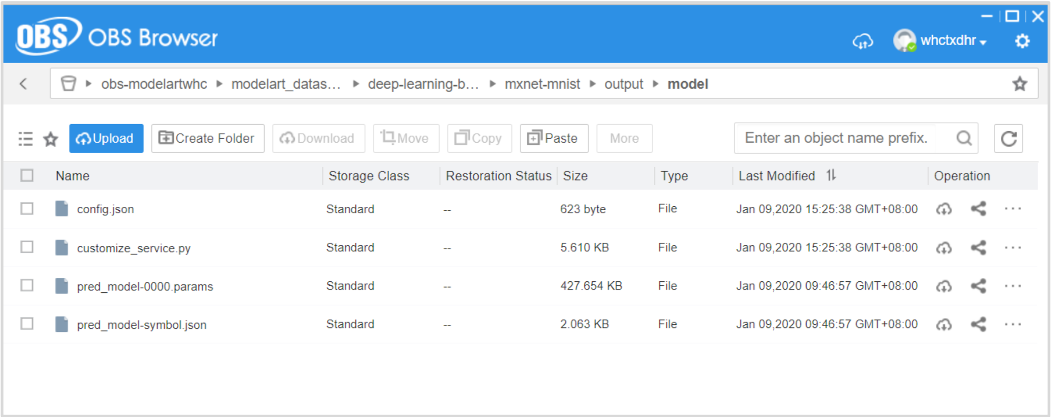


□ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6

Figure 4-17 Visualization job

The model directory contains the **pred_model-0000.params** and **pred_model-symbol.json** model files. This directory is used to import a model and deploy the model as a real-time service.

Step 5 Upload the **config.json** configuration file and **customize_service.py** inference code to the **model** folder in the OBS training output path, as shown in the following figure. Note that the configuration file name and inference code name cannot be changed.


OBS Browser
whctxdhr
obs-modelartwhc ▸ modelart_datas… ▸ deep-learning-b… ▸ mxnet-mnist ▸ output ▸ model
Upload
Create Folder
Download
Move
Copy
Paste
More
Enter an object name prefix.

| □ | Name | Storage Class | Restoration Status | Size | Type | Last Modified | Operation |
|---|---|---|---|---|---|---|---|
| □ | config.json | Standard | -- | 623 byte | File | Jan 09,2020 15:25:38 GMT+08:00 | ··· |
| □ | customize_service.py | Standard | -- | 5.610 KB | File | Jan 09,2020 15:25:38 GMT+08:00 | ··· |
| □ | pred_model-0000.params | Standard | -- | 427.654 KB | File | Jan 09,2020 09:46:57 GMT+08:00 | ··· |
| □ | pred_model-symbol.json | Standard | -- | 2.063 KB | File | Jan 09,2020 09:46:57 GMT+08:00 | ··· |

**Figure 4-18 model structure directory**

Interpretation of the **config.json** configuration file

```
"model_type":"MXNet",
# The fields in metrics are used to measure model accuracy. Their values range from 0 to 1. You can
set the fields to any value within this range.
"metrics": {"f1": 0.39542, "accuracy": 0.987426, "precision": 0.395875, "recall": 0.394966},
# Write the following code based on the object detection or image classification type. In this example,
the image classification type is used, and code is as follows:
# image_classification
"model_algorithm":"image_classification",
apis_dict['request'] = \
      {
        "data": {
    "type": "object",
    "properties": {
      "images": {
        "type": "file"
      }
    }
   },
   "Content-type": "multipart/form-data"
  }
    apis_dict['response'] = {
   "data": {
    "type": "object",
    "required": [
      "detection_classes",
      "detection_boxes",
      "detection_scores"
    ],
    "properties": {
      "detection_classes": {
        "type": "array",
        "item": {
          "type": "string"
        }
      },
      "detection_boxes": {
        "type": "array",
        "items": {
```

```
       "type": "array",
       "minItems": 4,
       "maxItems": 4,
       "items": {
        "type": "number"
       }
      }
     },
     "detection_scores": {
      "type": "number"
     }
    }
   },
   "Content-type": "multipart/form-data"
}
```

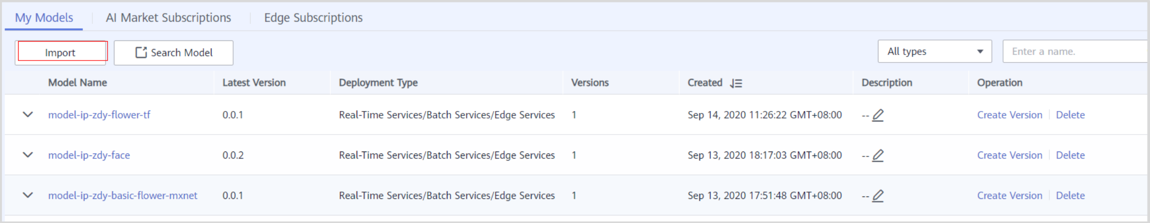
The following code is for object detection. The value of **model_algorithm** is **object_detection**.

```
"model_algorithm":"object_detection",
apis_dict['request'] = \
      {
        "data": {
    "type": "object",
    "properties": {
     "images": {
      "type": "file"
     }
    }
   },
   "Content-type": "multipart/form-data"
  }
    apis_dict['response'] = {
   "data": {
    "type": "object",
    "required": [
     "detection_classes",
     "detection_boxes",
     "detection_scores"
    ],
    "properties": {
     "detection_classes": {
      "type": "array",
      "item": {
       "type": "string"
      }
     },
     "detection_boxes": {
      "type": "array",
      "items": {
       "type": "array",
       "minItems": 4,
       "maxItems": 4,
       "items": {
        "type": "number"
       }
      }
      },
      "detection_scores": {
        "type": "number"
      }
    }
  },
  "Content-type": "multipart/form-data"
}
```

Interpretation of the **customize_service.py** inference code

```
# The built-in mxnet_model_service component of MXNet is used.
import mxnet as mx
import requests
import zipfile
import json
import shutil
import os
import numpy as np

from mxnet.io import DataBatch
from mms.log import get_logger
from mms.model_service.mxnet_model_service import MXNetBaseService
from mms.utils.mxnet import image, ndarray


logger = get_logger()
# Check whether the shape of the inputted image meets the requirements. If the shape does not
meet the requirements, an error is reported.
def check_input_shape(inputs, signature):
    '''Check input data shape consistency with signature.

    Parameters
    ----------
    inputs : List of NDArray
        Input data in NDArray format.
    signature : dict
        Dictionary containing model signature.
    '''
    assert isinstance(inputs, list), 'Input data must be a list.'
    assert len(inputs) == len(signature['inputs']), 'Input number mismatches with ' \
         'signature. %d expected but got %d.' \
                                            % (len(signature['inputs']), len(inputs))
    for input, sig_input in zip(inputs, signature['inputs']):
        assert isinstance(input, mx.nd.NDArray), 'Each input must be NDArray.'
        assert len(input.shape) == \
               len(sig_input['data_shape']), 'Shape dimension of input %s mismatches with ' \
                                'signature. %d expected but got %d.' \
                                % (sig_input['data_name'], len(sig_input['data_shape']),
                                   len(input.shape))
        for idx in range(len(input.shape)):
            if idx != 0 and sig_input['data_shape'][idx] != 0:
                assert sig_input['data_shape'][idx] == \
                       input.shape[idx], 'Input %s has different shape with ' \
                                         'signature. %s expected but got %s.' \
                                          % (sig_input['data_name'], sig_input['data_shape'],
                                             input.shape)
# Inherit the MXNetBaseService class. The MXNet model needs to inherit this base class when an
inference service is deployed.
class DLSMXNetBaseService(MXNetBaseService):
    '''MXNetBaseService defines the fundamental loading model and inference
       operations when serving MXNet model. This is a base class and needs to be
       inherited.
    '''
    def __init__(self, model_name, model_dir, manifest, gpu=None):
        print ("-------------------- init classification servive -------------")
        self.model_name = model_name
        self.ctx = mx.gpu(int(gpu)) if gpu is not None else mx.cpu()
        self._signature = manifest['Model']['Signature']
        data_names = []
        data_shapes = []
        for input in self._signature['inputs']:
            data_names.append(input['data_name'])
            # Replace 0 entry in data shape with 1 for binding executor.
            # Set batch size as 1
            data_shape = input['data_shape']
            data_shape[0] = 1
            for idx in range(len(data_shape)):
                if data_shape[idx] == 0:
                    data_shape[idx] = 1
            data_shapes.append(('data', tuple(data_shape)))

       # Load the MXNet model to the model directory of train_url. load_epoch of params can be
       # directly define here.
        epoch = 0
        try:
            param_filename = manifest['Model']['Parameters']
            epoch = int(param_filename[len(model_name) + 1: -len('.params')])
        except Exception as e:
            logger.warning('Failed to parse epoch from param file, setting epoch to 0')
       # load indicates the loaded well-trained model, and sym indicates model information,
including the contained layers. arg and aux are models.
       # Parameter information, which is stored in params on MXNet.
        sym, arg_params, aux_params = mx.model.load_checkpoint('%s/%s' % (model_dir,
manifest['Model']['Symbol'][:-12]), epoch)
       # Define a module, and place model network information and the contained parameters on
ctx, which can be a CPU or GPU.
        self.mx_model = mx.mod.Module(symbol=sym, context=self.ctx,
                                             data_names=['data'], label_names=None)
       # Bind the compute module to the compute engine.
        self.mx_model.bind(for_training=False, data_shapes=data_shapes)
       # Set the parameter to the parameter of the trained model.
        self.mx_model.set_params(arg_params, aux_params, allow_missing=True)
   # Read images and data. The function is called when its name contains _preprocess.
    def _preprocess(self, data):
        img_list = []
        for idx, img in enumerate(data):
            input_shape = self.signature['inputs'][idx]['data_shape']
            # We are assuming input shape is NCHW
            [h, w] = input_shape[2:]
            if input_shape[1] == 1:
                img_arr = image.read(img, 0)
            else:
                img_arr = image.read(img)
            # Resize the image to 28 x 28 pixels.
            img_arr = image.resize(img_arr, w, h)
            # Re-arrange the image to the NCHW format.
            img_arr = image.transform_shape(img_arr)
            img_list.append(img_arr)
        return img_list
     # Summarize the inference results, and return top 5 confidence.
    def _postprocess(self, data):
        dim = len(data[0].shape)
        if dim > 2:
            data = mx.nd.array(np.squeeze(data.asnumpy(), axis=tuple(range(dim)[2:])))
        sorted_prob = mx.nd.argsort(data[0], is_ascend=False)
       # Define the output as top 5.
        top_prob = map(lambda x: int(x.asscalar()), sorted_prob[0:5])
        return [{'probability': float(data[0, i].asscalar()), 'class': i}
                for i in top_prob]
   # Perform a forward process to obtain the model result output.
    def _inference(self, data):
        '''Internal inference methods for MXNet. Run forward computation and
        return output.

        Parameters
        ----------
        data : list of NDArray
            Preprocessed inputs in NDArray format.

        Returns
        -------
        list of NDArray
            Inference output.
        '''
       # Check the data format.
        check_input_shape(data, self.signature)
        data = [item.as_in_context(self.ctx) for item in data]
        self.mx_model.forward(DataBatch(data))
        return self.mx_model.get_outputs()[0]
   # The ping and signature functions are used to check whether the service is normal. You can
define the functions as follows:
    def ping(self):
        '''Ping to get system's health.

        Returns
        -------
        String
            MXNet version to show system is healthy.
        '''
        return mx.__version__

    @property
    def signature(self):
        '''Signiture for model service.
        Returns
        -------
        Dict
            Model service signiture.
        '''
        return self._signature
```

Step 6 Import a model and deploy it as a real-time prediction service. In the navigation pane, click **Model Management**. On the displayed page, click **Import**. See the following figure.



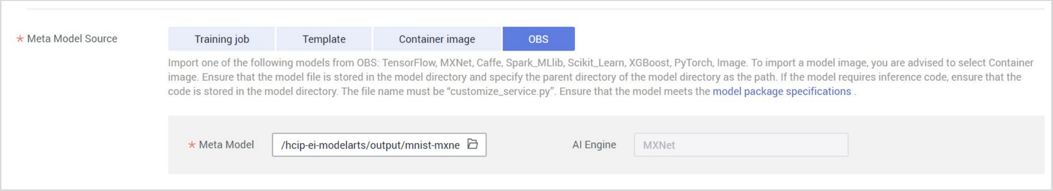
My Models | AI Market Subscriptions | Edge Subscriptions

Import | Search Model | All types | Enter a name.

| | Model Name | Latest Version | Deployment Type | Versions | Created | Description | Operation |
|---|---|---|---|---|---|---|---|
| ∨ | model-ip-zdy-flower-tf | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 14, 2020 11:26:22 GMT+08:00 | -- | Create Version \| Delete |
| ∨ | model-ip-zdy-face | 0.0.2 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 13, 2020 18:17:03 GMT+08:00 | -- | Create Version \| Delete |
| ∨ | model-ip-zdy-basic-flower-mxnet | 0.0.1 | Real-Time Services/Batch Services/Edge Services | 1 | Sep 13, 2020 17:51:48 GMT+08:00 | -- | Create Version \| Delete |

**Figure 4-19 Importing a model**

Select the path of the specified meta model. When selecting the path, select the upper-level directory of the **model** file and click **Create Now**. See the following figure.



* Meta Model Source: Training job | Template | Container image | OBS

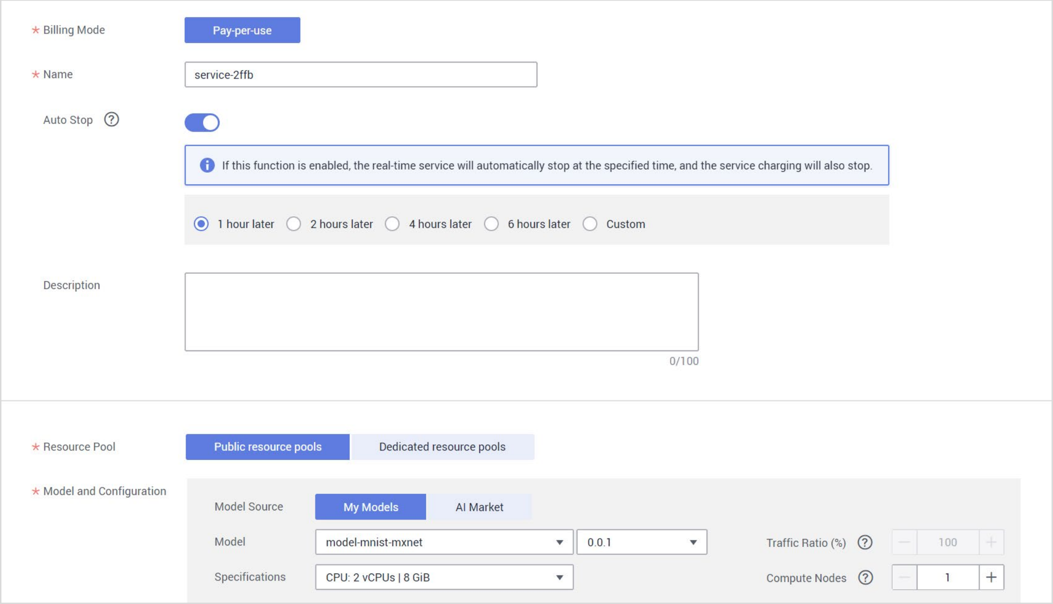
Import one of the following models from OBS: TensorFlow, MXNet, Caffe, Spark_MLlib, Scikit_Learn, XGBoost, PyTorch, Image. To import a model image, you are advised to select Container image. Ensure that the model file is stored in the model directory and specify the parent directory of the model directory as the path. If the model requires inference code, ensure that the code is stored in the model directory. The file name must be "customize_service.py". Ensure that the model meets the model package specifications .

* Meta Model: /hcip-ei-modelarts/output/mnist-mxne

AI Engine: MXNet

**Figure 4-20 Selecting a path for importing a model**

On the Model Management page, locate the mx_mnist_demo model and choose Deploy > Real-Time Services.



* Billing Mode: Pay-per-use

* Name: service-2ffb

Auto Stop ⓘ

If this function is enabled, the real-time service will automatically stop at the specified time, and the service charging will also stop.

● 1 hour later ○ 2 hours later ○ 4 hours later ○ 6 hours later ○ Custom

Description

0/100

* Resource Pool: Public resource pools | Dedicated resource pools

* Model and Configuration

Model Source: My Models | AI Market

Model: model-mnist-mxnet | 0.0.1

Traffic Ratio (%) ⓘ 100

Specifications: CPU: 2 vCPUs | 8 GiB

Compute Nodes ⓘ 1

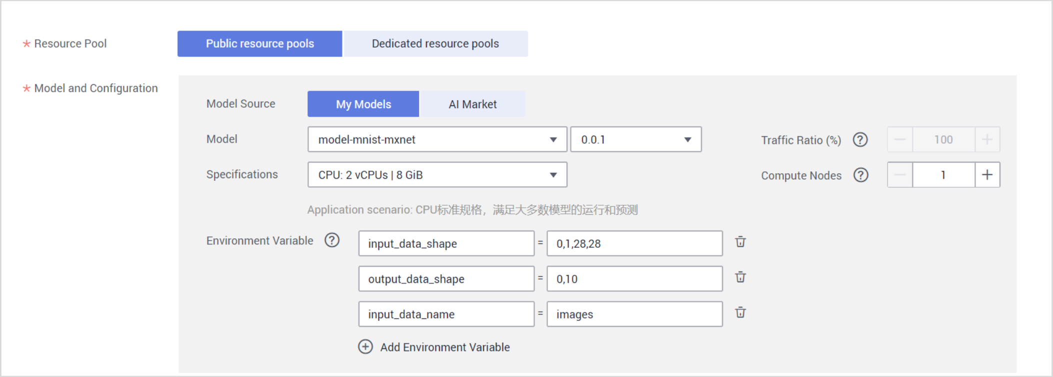
## Figure 4-21 Deploying a real-time service

On the **Deploy** page, enter the following parameters:

**input_data_shape** indicates the shape of the inputted image. The MNIST dataset contains 28 x 28 pixels images. Therefore, enter **0,1,28,28**.

**output_data_shape** is the model output. MNIST is a sample set of 10 classes. Therefore, enter **0,10**, which indicates a value ranging from 0 to 10.

**input_data_name** is set to **images** for tests on the public cloud UI. If the API structure is invoked, this parameter can be left blank. See the following figure.



* Resource Pool: Public resource pools | Dedicated resource pools

* Model and Configuration

Model Source: My Models | AI Market

Model: model-mnist-mxnet | 0.0.1

Traffic Ratio (%): 100

Specifications: CPU: 2 vCPUs | 8 GiB

Compute Nodes: 1

Application scenario: CPU标准规格，满足大多数模型的运行和预测

Environment Variable:

input_data_shape = 0,1,28,28

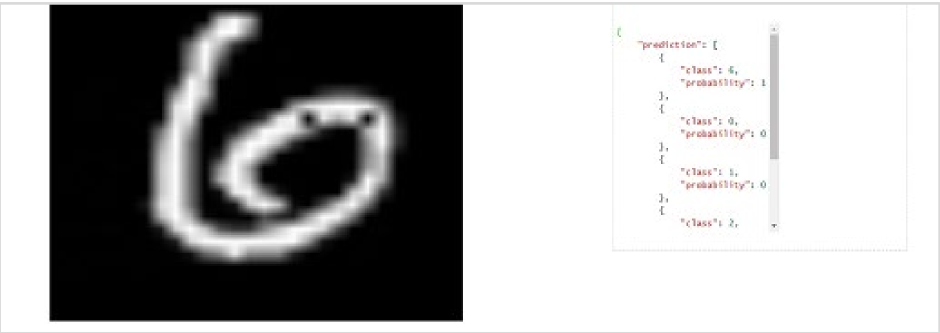
output_data_shape = 0,10

input_data_name = images

Add Environment Variable

## Figure 4-22 Deploying a real-time service

After the service deployment is complete, upload the image in the following path: modelarts-datasets-and-source-code/custom-basic-algorithms-for-deep learning/native-MXNet-for-handwritten-digit-recognition/test-data/6.jpg. The 28 x 28 pixels MNIST handwritten images with white characters on black background are used for testing. See the following figure.



```
{
    "prediction": [
        {
            "class": 6,
            "probability": 1
        },
        {
            "class": 0,
            "probability": 0
        },
        {
            "class": 1,
            "probability": 0
        },
        {
            "class": 2,
```

## Figure 4-23 Test result of the real-time service

NOTE

After the experiment is complete, disable the service in a timely manner to avoid unnecessary expenses.