Huawei AI Certification Training

HCIA-AI

ModelArts Experiment Guide

ISSUE:3.0

HUAWEI

HUAWEI TECHNOLOGIES CO., LTD.

Huawei Technologies Co., Ltd.

Address:	Huawei Industrial Base Bantian, Longgang Shenzhen 518129 People's Republic of China
Website:	http://e.huawei.com

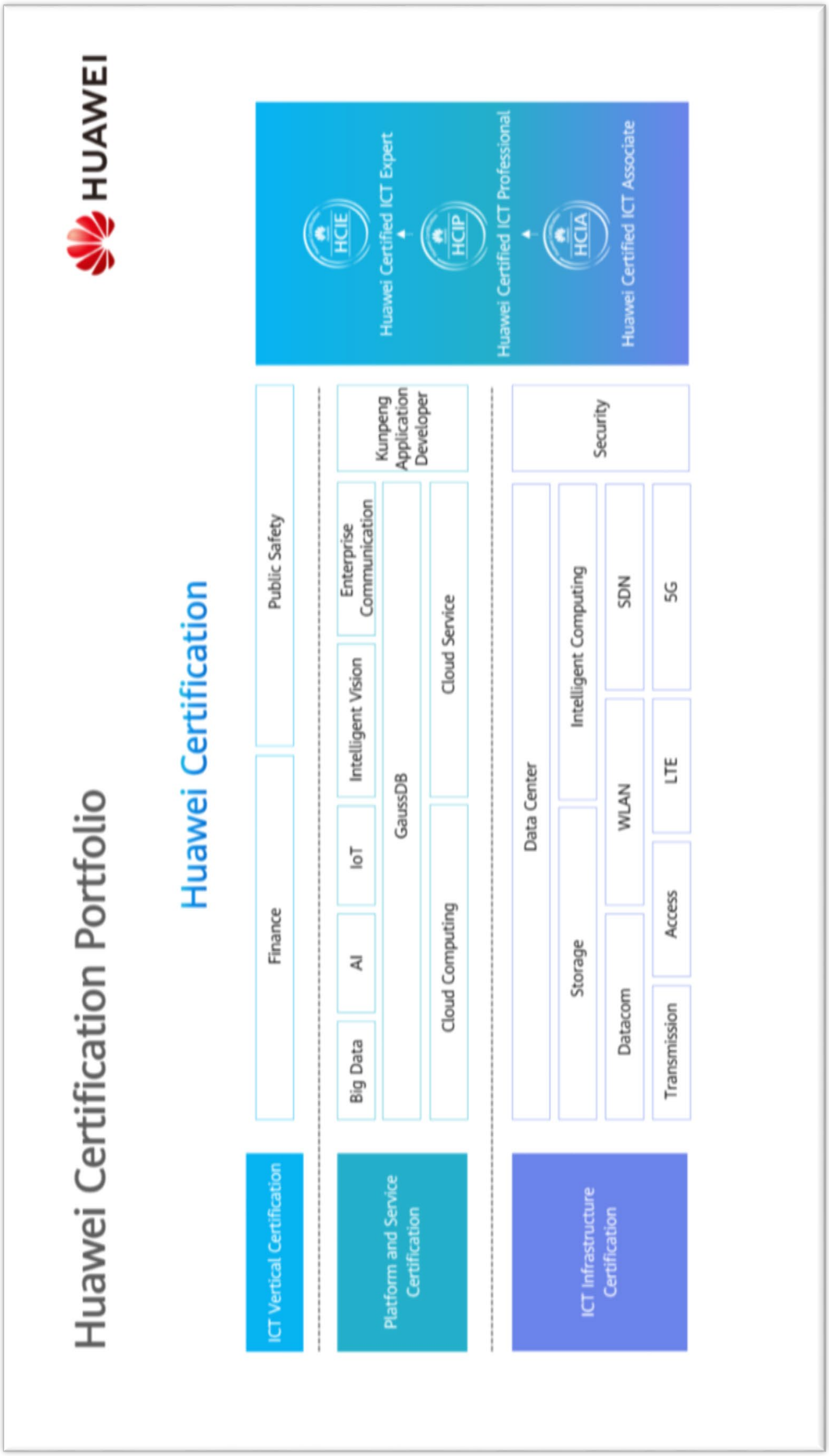
Huawei Certificate System

Huawei Certification follows the "platform + ecosystem" development strategy, which is a new collaborative architecture of ICT infrastructure based on "Cloud-Pipe-Terminal". Huawei has set up a complete certification system consisting of three categories: ICT infrastructure certification, Platform and Service certification and ICT vertical certification, and grants Huawei certification the only all-range technical certification in the industry.

Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

HCIA-AI V3.0 aims to train and certify engineers who are capable of designing and developing AI products and solutions using algorithms such as machine learning and deep learning.

HCIA-AI V3.0 certification demonstrates that: You know the development history of AI, Huawei Ascend AI system and full-stack all-scenario AI strategies, and master traditional machine learning and deep learning algorithms; you can use the TensorFlow and MindSpore development frameworks to build, train, and deploy neural networks; you are competent for sales, marketing, product manager, project management, and technical support positions in the AI field.



Huawei Certification Portfolio

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
- Storage
- Intelligent Computing
- Security
- 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 applicable to the candidates who are preparing for the HCIA-AI exam and the readers who want to understand the AI programming basics. After learning this guide, you will be able to perform basic AI programming with ModelArts.

Description

This guide contains two experiments, which are based on how to use ModelArts. It is hoped that trainees or readers can get started with deep learning and have the basic programming capability by ModelArts.

If you are in an area where Huawei Cloud and ModelArts service is not accessible, this chapter can be ignored, it is not necessary for passing the HCIA-AI V3.0 certification.

Background Knowledge Required

To fully understand this course, the readers should have basic Python programming capabilities, knowledge of data structures and deep learning algorithms.

Experiment Environment Overview

Huawei Cloud and ModelArts

Contents

1 ExeML

1.1 Introduction

1.1.1 About This Experiment

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.

1.1.2 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.

1.1.3 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. The following picture shows the steps.

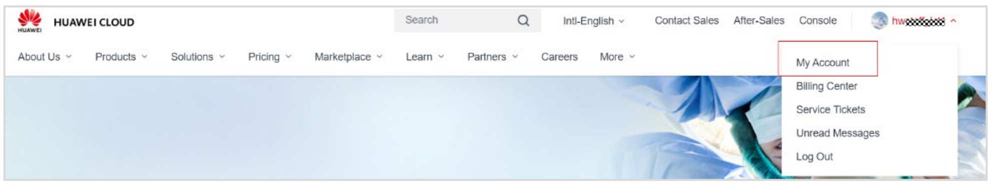
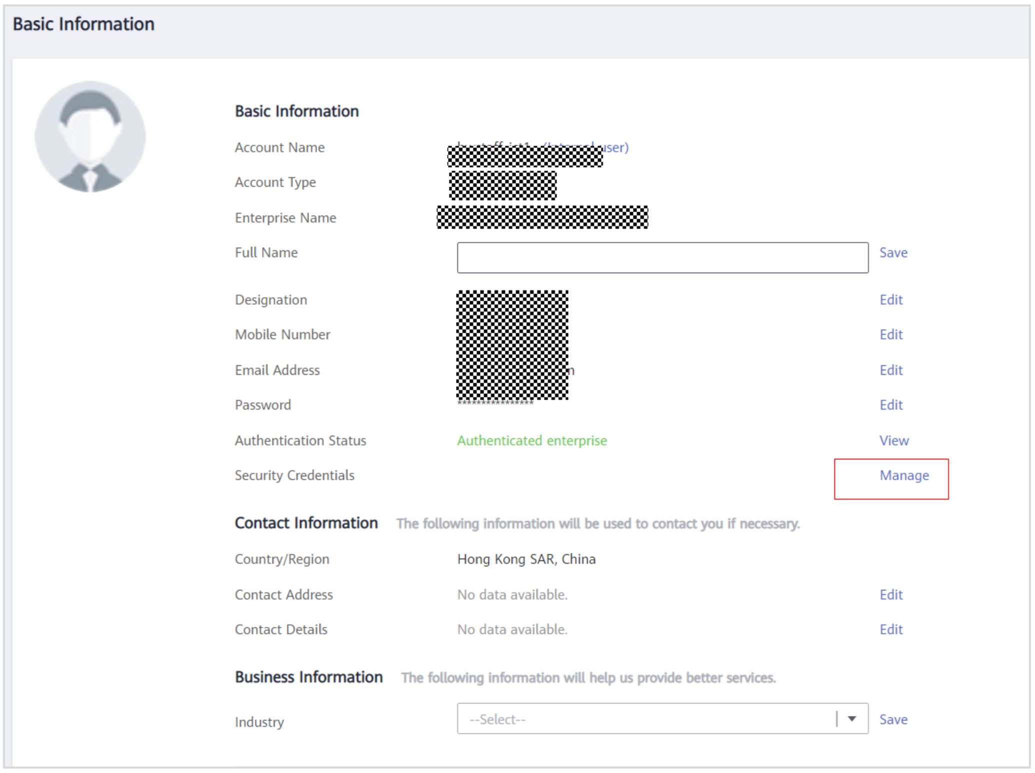


Figure 1-1 Huawei Cloud Homepage



Basic Information

Basic Information

Account Name	[redacted]	
Account Type	[redacted]	
Enterprise Name	[redacted]	
Full Name		Save
Designation	[redacted]	Edit
Mobile Number		Edit
Email Address		Edit
Password		Edit
Authentication Status	Authenticated enterprise	View
Security Credentials		Manage

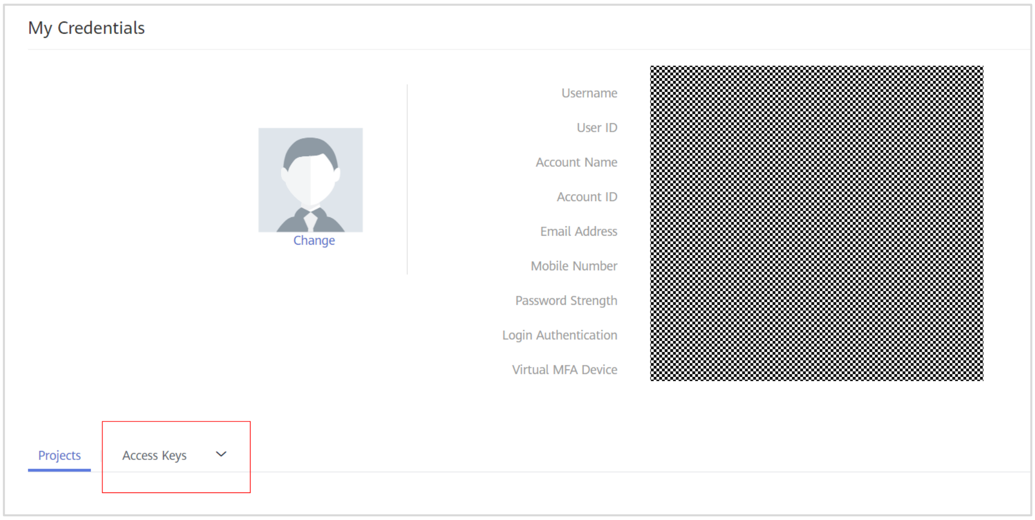
Contact Information The following information will be used to contact you if necessary.

Country/Region	Hong Kong SAR, China	
Contact Address	No data available.	Edit
Contact Details	No data available.	Edit

Business Information The following information will help us provide better services.

Industry	--Select--	Save

Figure 1-2 Basic Information Page



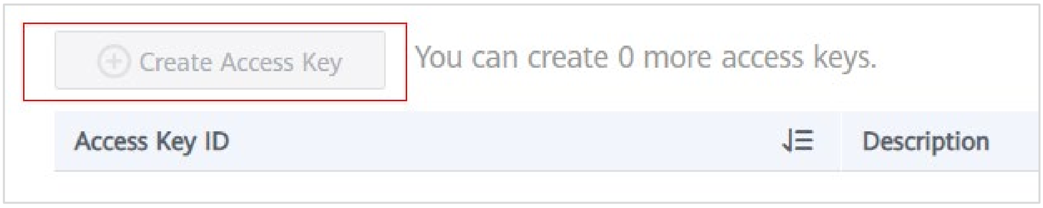
My Credentials

Change

Username	[redacted]
User ID	
Account Name	
Account ID	
Email Address	
Mobile Number	
Password Strength	
Login Authentication	
Virtual MFA Device	

Projects Access Keys

Figure 1-3 My Credentials Page



Create Access Key You can create 0 more access keys.

Access Key ID	Description

Figure 1-4 Create Access Key

- 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.

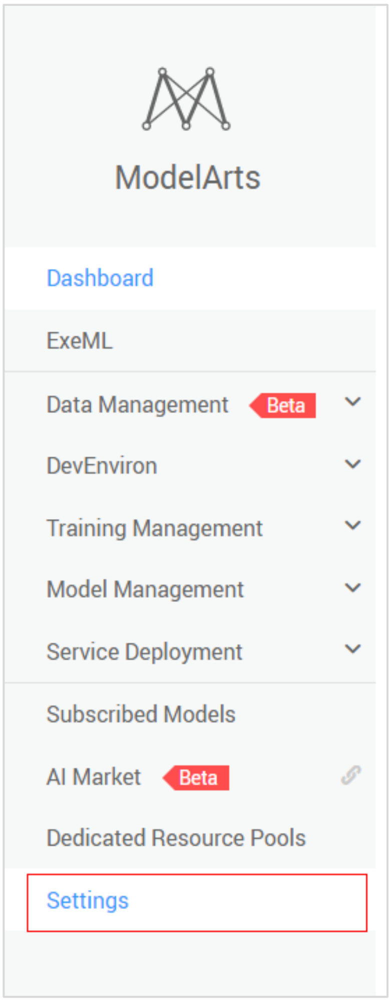


Figure 1-5 ModelArts Settings Page

1.2 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.

1.2.1 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.

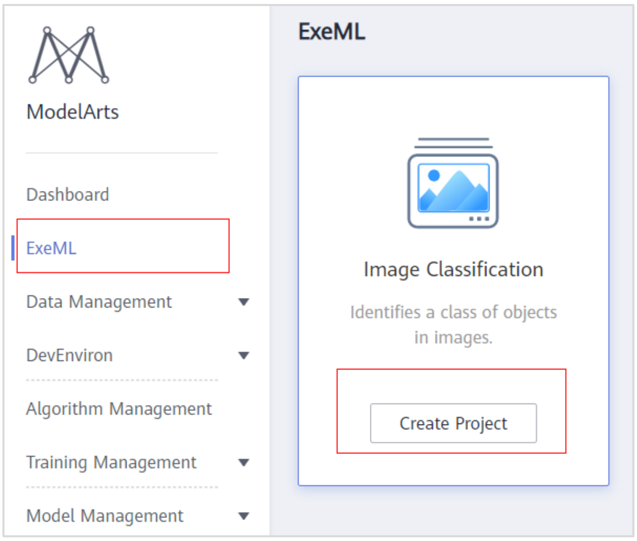
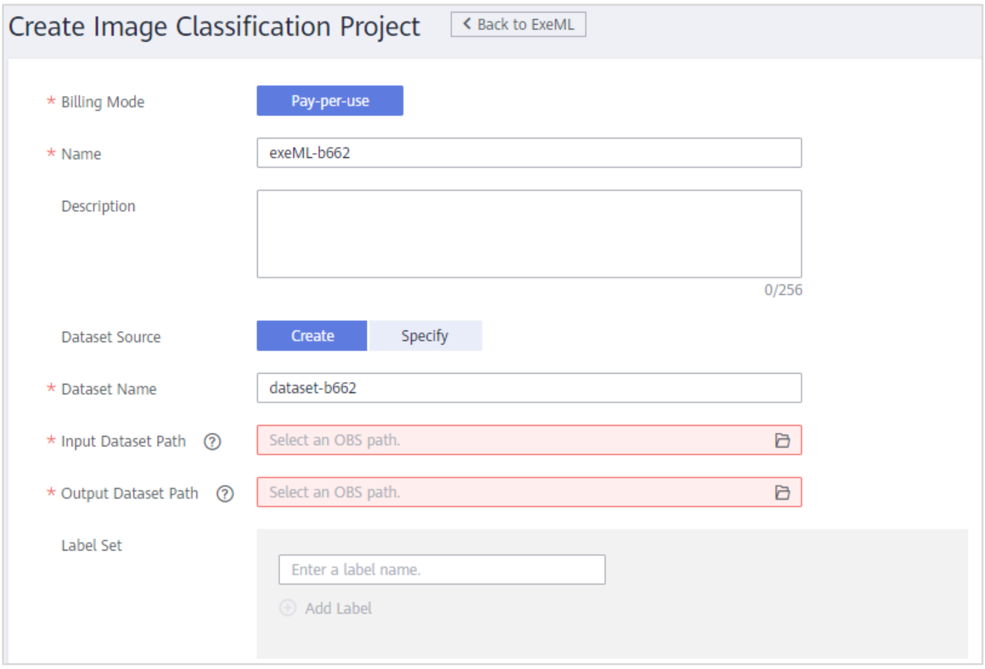


Figure 1-6 ModelArts ExeML



Create Image Classification Project
Back to ExeML

* Billing Mode	Pay-per-use
* Name	exeML-b662
Description	0/256
Dataset Source	Create / Specify
* Dataset Name	dataset-b662
* Input Dataset Path	Select an OBS path.
* Output Dataset Path	Select an OBS path.
Label Set	Enter a label name. Add Label

Figure 1-7 Create Image Classification Project

Parameters:

- **Billing Mode: Pay-per-use** by default
- **Name:** The value can be modified as required.
- **Description:** The value can be modified as required.
- **Dataset 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: https://data-certification.obs.cn-east-2.myhuaweicloud.com/ENG/HCIA-AI/V3.0/modelarts-demo.rar
- **Output Dataset Path:** Select an OBS path for storing the dataset to be exported. Create an empty folder on OBS, the method of creation is the same as **Input Dataset Path**.

Step 2 Confirm the project creation.

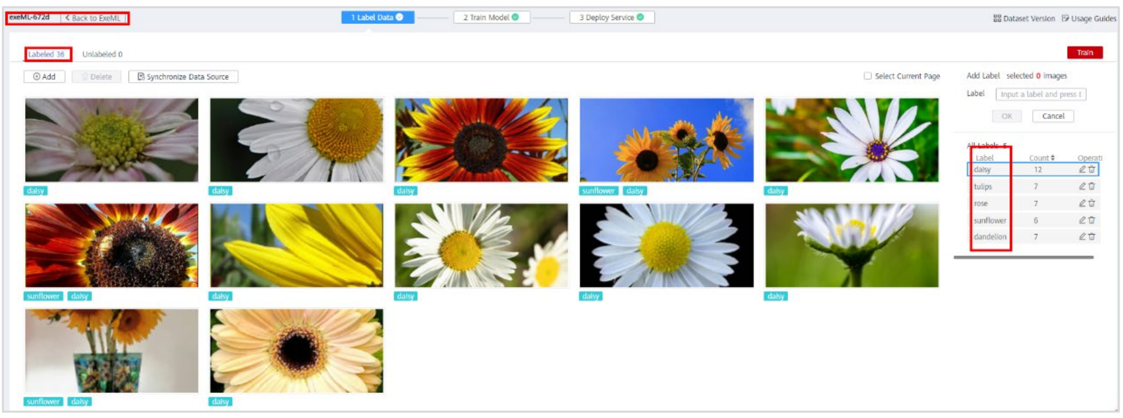
Click **Create Project**. The ExeML project is created.

1.2.2 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.



Label	Count	Operati
daisy	12	
tulips	7	
rose	7	
sunflower	6	
dandelion	7	

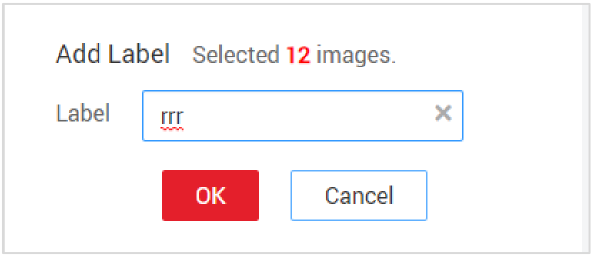
Figure 1-8 Upload Images

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.



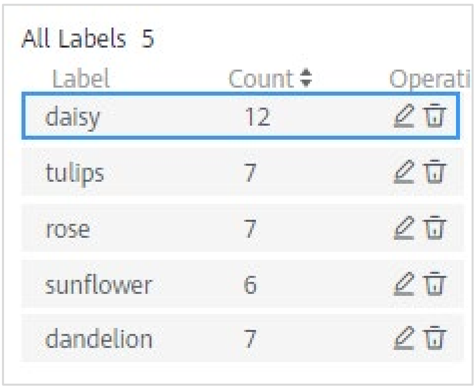
Add Label Selected 12 images.

Label: rrr

Figure 1-9 Label the Images

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 1-10.



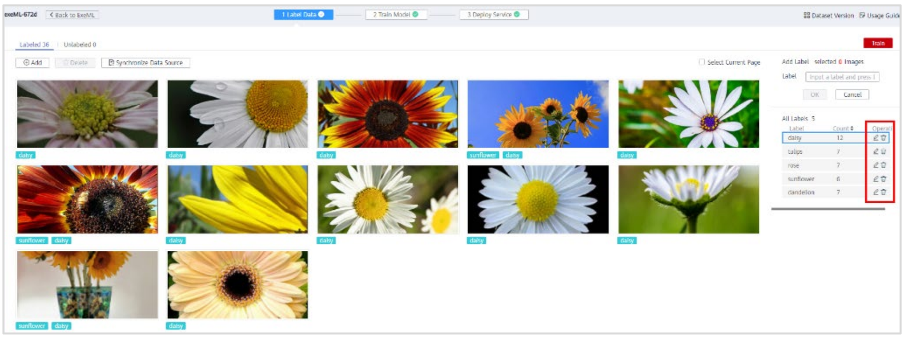
All Labels 5

Label	Count	Operati
daisy	12	✎ 🗑
tulips	7	✎ 🗑
rose	7	✎ 🗑
sunflower	6	✎ 🗑
dandelion	7	✎ 🗑

Figure 1-10 Delete or Modify 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.**



All Labels 5

Label	Count	Operation
daisy	12	
tulips	7	
rose	7	
sunflower	6	
dandelion	7	

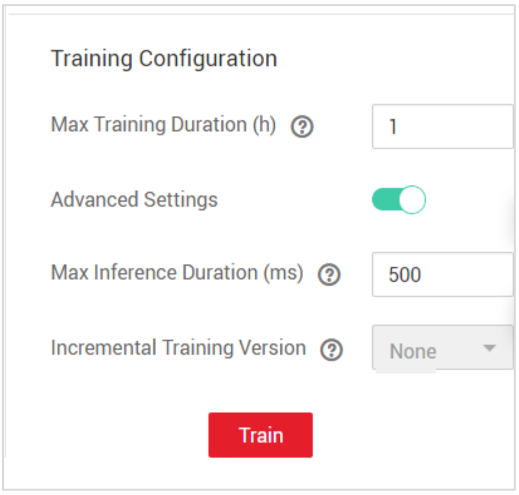
Figure 1-11 Delete or Modify a Label in Multiple Images

1.2.3 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 1-12 shows the training settings.


Training Configuration

Max Training Duration (h): 1

Advanced Settings: on

Max Inference Duration (ms): 500

Incremental Training Version: None

Figure 1-12 Training Configuration

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.

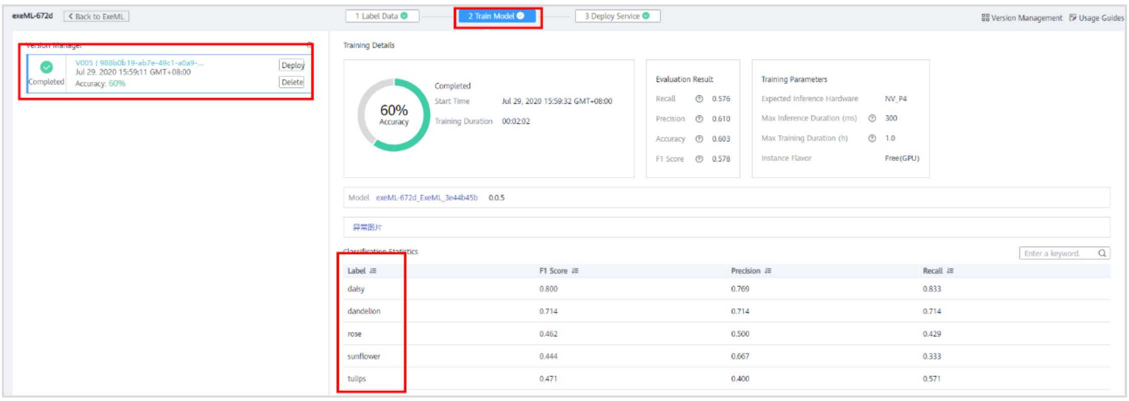
Step 2 Train a model.

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

1.2.4 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 1-13. After the deployment is successful, you can choose **Service Deployment > Real-Time Services** to view the deployed service.



V005 (988b0b19-ab7e-49c1-a0a9-...
Completed
Jul 29. 2020 15:59:11 GMT+08:00
Accuracy: 60%

Completed
Start Time Jul 29, 2020 15:59:32 GMT+08:00
Training Duration 00:02:02

Evaluation Result

Recall	0.576
Precision	0.610
Accuracy	0.603
F1 Score	0.578

Training Parameters

Expected Inference Hardware	NV_P4
Max Inference Duration (ms)	300
Max Training Duration (h)	1.0
Instance Flavor	Free(GPU)

Model exeML-672d_ExeML_3e44b45b 0.0.5

Classification Statistics

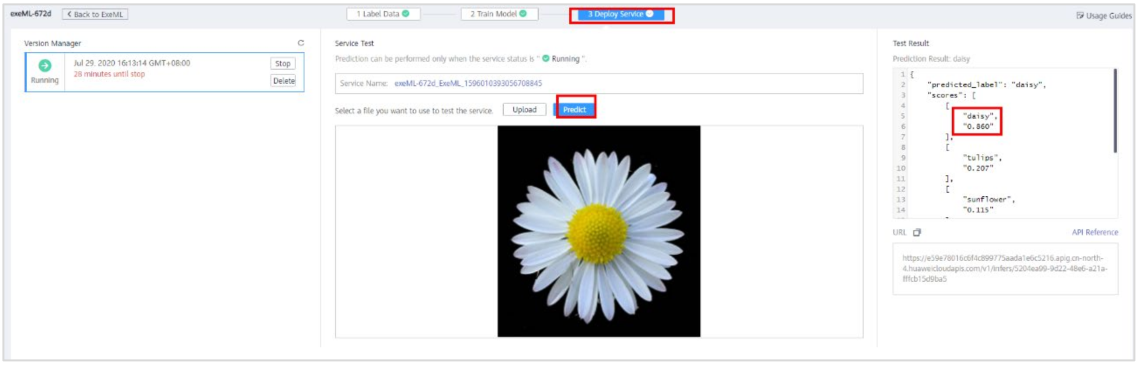
Label	F1 Score	Precision	Recall
daisy	0.800	0.769	0.833
dandelion	0.714	0.714	0.714
rose	0.462	0.500	0.429
sunflower	0.444	0.667	0.333
tulips	0.471	0.400	0.571

Figure 1-13 Train Model

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 1-14. 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".



Jul 29, 2020 16:13:14 GMT+08:00
28 minutes until stop

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

Service Name: exeML-672d_ExeML_1596010393056708845

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

Prediction Result: daisy

```
{
    "predicted_label": "daisy",
    "scores": [
        [
            "daisy",
            "0.860"
        ],
        [
            "tulips",
            "0.207"
        ],
        [
            "sunflower",
            "0.115"
```

https://e59e78016c6f4c899775aada1e6c5216.apig.cn-north-4.huaweicloudapis.com/v1/infers/5204ea99-9d22-48e6-a21a-fffcb15d9ba5

Figure 1-14 Deploy Service

2 Image Classification

2.1 Introduction

2.1.1 About This Experiment

This experiment is about a basic task in computer vision, that is image recognition. This experiment has been completed in chapter 4.Now, we will do this experiment on ModelArts with GPU.

2.1.2 Objectives

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

- Know how to use ModelArts to develop your own model
- Understand the advantages of deep neural network training with GPU acceleration

2.2 Procedure

2.2.1 Create a Notebook

Step 1 Create a Notebook

Open the **ModelArts Homepage** and select **DevEnviron** to create Notebook.

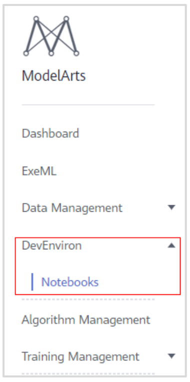
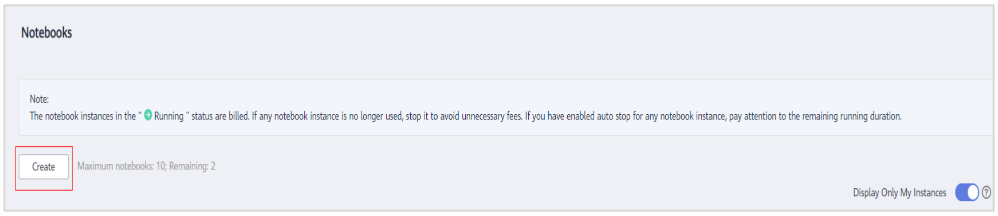


Figure 2-1 ModelArts Homepage

Click the Create button to Create the Notebook environment.


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.

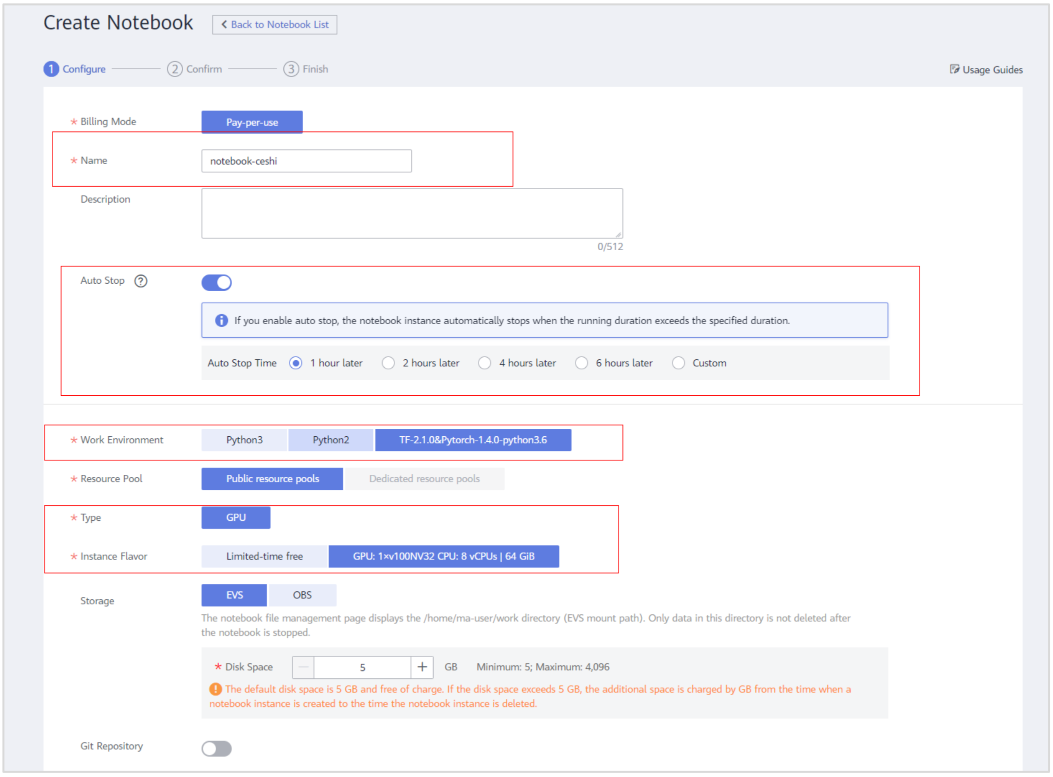
Create
Maximum notebooks: 10; Remaining: 2
Display Only My Instances

Figure 2-2 Create Notebook

Step 2 Configuration

Clicking the Create button, then you got the configuration page.Here are some parameters, explained as follows:

- **Name:** user-defined.
- **Auto Stop:** user-defined, it is recommended to estimate the usage time in advance.
- **Work Environment:** Here, TF2.1 is fixed, as shown in the **figure 2-3.**
- **Instance Flavor: GPU:** 1xV100 is recommended.



Create Notebook < Back to Notebook List

1 Configure —— 2 Confirm —— 3 Finish Usage Guides

* Billing Mode: Pay-per-use

* Name: notebook-ceshi

Description: 0/512

Auto Stop: on

If you enable auto stop, the notebook instance automatically stops when the running duration exceeds the specified duration.

Auto Stop Time: [✓ 1 hour later] 2 hours later 4 hours later 6 hours later Custom

* Work Environment: Python3 Python2 [✓ TF-2.1.0&Pytorch-1.4.0-python3.6]

* Resource Pool: [✓ Public resource pools] Dedicated resource pools

* Type: [✓ GPU]

* Instance Flavor: Limited-time free [✓ GPU: 1×v100NV32 CPU: 8 vCPUs | 64 GiB]

Storage: [✓ EVS] OBS

The notebook file management page displays the /home/ma-user/work directory (EVS mount path). Only data in this directory is not deleted after the notebook is stopped.

* Disk Space: 5 GB Minimum: 5; Maximum: 4,096

The default disk space is 5 GB and free of charge. If the disk space exceeds 5 GB, the additional space is charged by GB from the time when a notebook instance is created to the time the notebook instance is deleted.

Git Repository: off

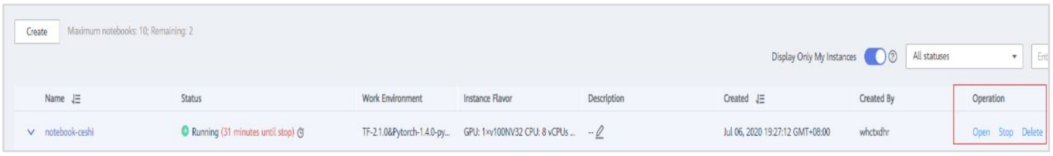
Figure 2-3 Notebook Configuration

Step 3 Finish

Then, at the bottom of this page, click Next button to finish creating notebook.

2.2.2 Open Notebook and Coding

Once the Notebook is created, the Notebook environment that you created will be listed on this page. The Notebook environment can be managed with the Open, Stop, and Delete buttons. Click Open to continue.



Create Maximum notebooks: 10; Remaining: 2

Display Only My Instances All statuses

Name	Status	Work Environment	Instance Flavor	Description	Created	Created By	Operation
notebook-ceshi	Running (31 minutes until stop)	TF-2.1.0&Pytorch-1.4.0-py...	GPU: 1×v100NV32 CPU: 8 vCPUs ...	--	Jul 06, 2020 19:27:12 GMT+08:00	whctxdhr	Open Stop Delete

Figure 2-4 Open Notebook

This interface is a Notebook page, click the **New** button, and then click **TensorFlow-2.1.0** to create a New file, as shown on this figure.

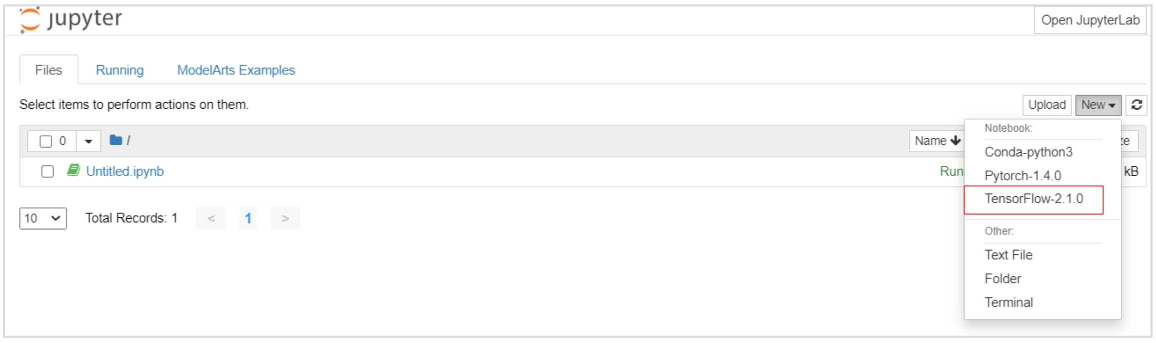
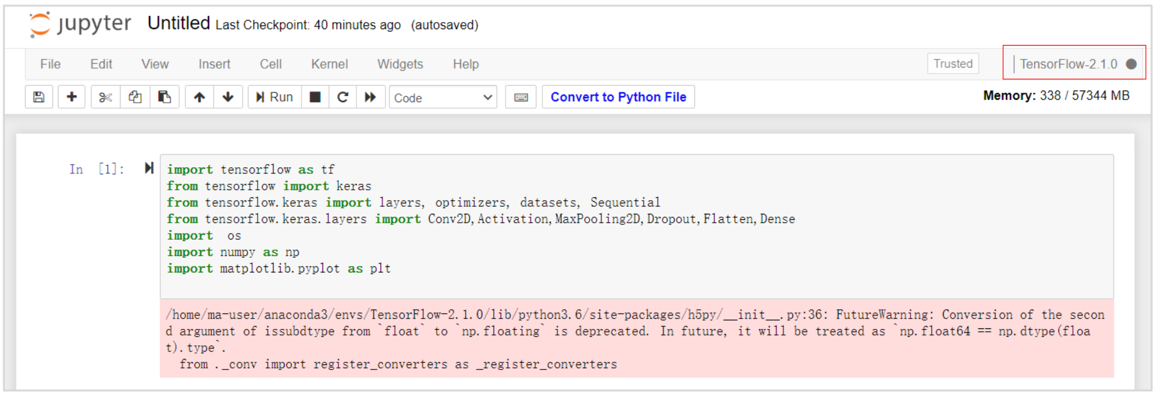


Figure 2-5 Create a New File

This page is a Notebook's code editing interface and its base environment is TensorFlow2.1.0. You can start editing the code.



```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, optimizers, datasets, Sequential
from tensorflow.keras.layers import Conv2D,Activation,MaxPooling2D,Dropout,Flatten,Dense
import  os
import numpy as np
import matplotlib.pyplot as plt
```

```
/home/ma-user/anaconda3/envs/TensorFlow-2.1.0/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the secon
d argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(floa
t).type`.
  from ._conv import register_converters as _register_converters
```

Figure 2-6 Notebook's Code Editing Interface

2.3 Experiment Code

2.3.1 Introducing Dependencies

Code:

```
import imageio
import numpy as np
import pickle
import os
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, optimizers, datasets, Sequential
from tensorflow.keras.layers import Conv2D,Activation,MaxPooling2D,Dropout,Flatten,Dense
```

2.3.2 Data Preprocessing

Code:

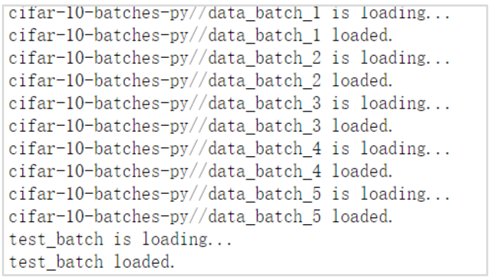
```
!wget https://data-certification.obs.cn-east-2.myhuaweicloud.com/ENG/HCIA-AI/V3.0/cifar-10-python.tar.gz
```

```
!tar -zxvf cifar-10-python.tar.gz
```

Code:

```
final_tr_data=[]
final_tr_label=[]
final_te_data=[]
final_te_label=[]

# Decompress, return the decompressed dictionary
def unpickle(file):
    fo = open(file, 'rb')
    dict = pickle.load(fo, encoding='latin1')
    fo.close()
    return dict

# Generate training set pictures. If you need png format, you only need to change the picture suffix name.
for j in range(1, 6):
    dataName = "cifar-10-batches-py//data_batch_" + str(j)   # Read the data_batch12345 file in the current directory. The dataName is actually the path of the data_batch file. The text and the script file are in the same directory.
    Xtr = unpickle(dataName)
    print(dataName + " is loading...")

    for i in range(0, 10000):
        img = np.reshape(Xtr['data'][i], (3,32, 32))   # Xtr['data'] is picture binary data
        img = img.transpose(1, 2, 0)
        picName = 'train//' + str(Xtr['labels'][i]) + '_' + str(i + (j - 1)*10000) + '.jpg'   # Xtr['labels'] is the label of the picture, the value range is 0-9. In this article, the train folder needs to exist and be in the same directory as the script file.
        if not os.path.exists('./train'):
            os.mkdir('./train')

        imageio.imwrite(picName, img)
        final_tr_data.append(img)
        final_tr_label.append(Xtr['labels'][i])

    print(dataName + " loaded.")

print("test_batch is loading...")

# Generate test set images
testXtr = unpickle("cifar-10-batches-py//test_batch")
for i in range(0, 10000):
    img = np.reshape(testXtr['data'][i], (3,32, 32))
    img = img.transpose(1, 2, 0)

    picName = 'test//' + str(testXtr['labels'][i]) + '_' + str(i) + '.jpg'
    if not os.path.exists('./test'):
        os.mkdir('./test')
    imageio.imwrite(picName, img)
    final_te_data.append(img)
    final_te_label.append(testXtr['labels'][i])
print("test_batch loaded.")
```

Output：

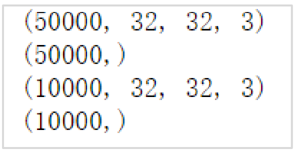


```
cifar-10-batches-py//data_batch_1 is loading...
cifar-10-batches-py//data_batch_1 loaded.
cifar-10-batches-py//data_batch_2 is loading...
cifar-10-batches-py//data_batch_2 loaded.
cifar-10-batches-py//data_batch_3 is loading...
cifar-10-batches-py//data_batch_3 loaded.
cifar-10-batches-py//data_batch_4 is loading...
cifar-10-batches-py//data_batch_4 loaded.
cifar-10-batches-py//data_batch_5 is loading...
cifar-10-batches-py//data_batch_5 loaded.
test_batch is loading...
test_batch loaded.
```

Code：

```
final_tr_data=np.array(final_tr_data)
print(np.shape(final_tr_data))
y_train=np.array(final_tr_label).reshape(50000,1)
print(np.shape(final_tr_label))

final_te_data=np.array(final_te_data)
print(np.shape(final_te_data))
y_test=np.array(final_te_label).reshape(10000,1)
print(np.shape(final_te_label))
```

Output：



```
(50000, 32, 32, 3)
(50000,)
(10000, 32, 32, 3)
(10000,)
```

Show the first 9 images:

Code:

```
category_dict = {0:'airplane',1:'automobile',2:'bird',3:'cat',4:'deer',5:'dog',
                  6:'frog',7:'horse',8:'ship',9:'truck'}
#Show the first 9 images and their labels
plt.figure()
for i in range(9):
    #create a figure with 9 subplots
    plt.subplot(3,3,i+1)
    #show an image
    plt.imshow(final_tr_data[i])
    #show the label
    plt.ylabel(category_dict[final_tr_label[i]])
plt.show()

#Pixel normalization
x_train = final_tr_data.astype('float32')/255
x_test = final_te_data.astype('float32')/255

num_classes=10
```

Output:

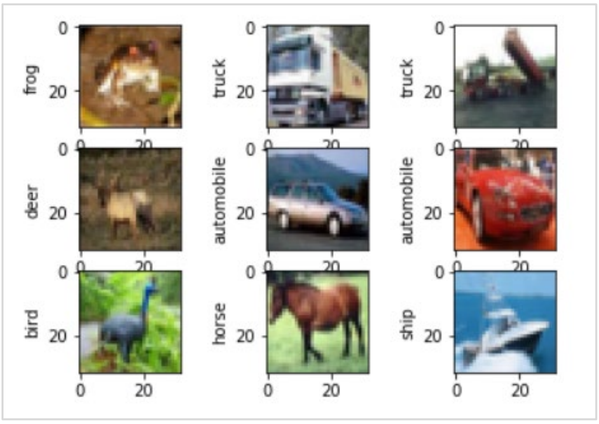


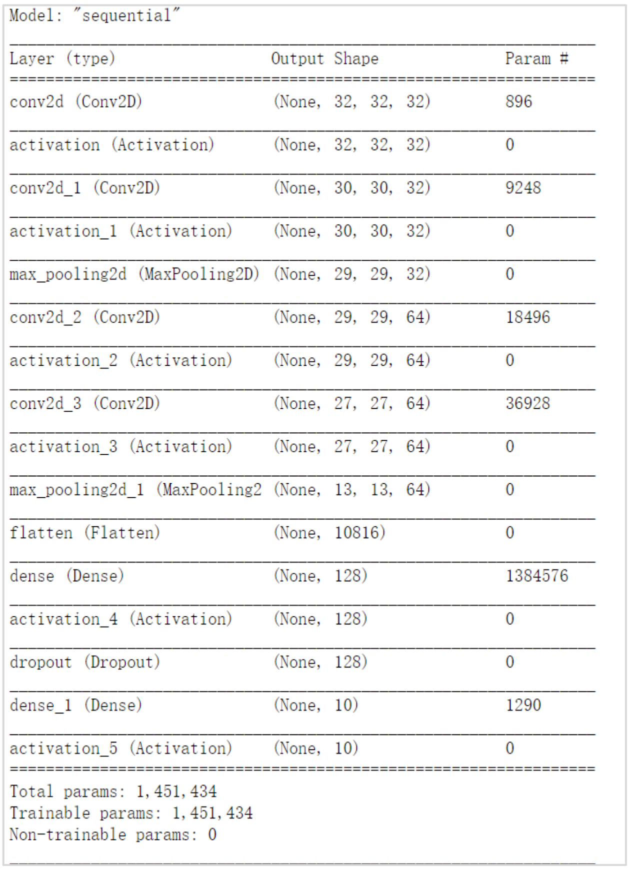
Figure 2-7 First 9 Images with Tags

2.3.3 Model Creation

Code:

```
def CNN_classification_model(input_size = x_train.shape[1:]):
    model = Sequential()
    #the first block with 2 convolutional layers and 1 maxpooling layer
    '''Conv1 with 32 3*3 kernels
        padding="same": it applies zero padding to the input image so that the input image gets
fully covered by the filter and specified stride.
        It is called SAME because, for stride 1 , the output will be the same as the input.
        output: 32*32*32'''
    model.add(Conv2D(32, (3, 3), padding='same',
              input_shape=input_size))
    #relu activation function
    model.add(Activation('relu'))
    #Conv2
    model.add(Conv2D(32, (3, 3)))
    model.add(Activation('relu'))
    #maxpooling
    model.add(MaxPooling2D(pool_size=(2, 2),strides =1))

    #the second block
    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    #maxpooling.the default strides =1
    model.add(MaxPooling2D(pool_size=(2, 2)))


    #Before sending a feature map into a fully connected network, it should be flattened into a
column vector.
    model.add(Flatten())
    #fully connected layer
    model.add(Dense(128))
    model.add(Activation('relu'))
    #dropout layer.every neuronis set to 0 with a probability of 0.25
    model.add(Dropout(0.25))
    model.add(Dense(num_classes))
    #map the score of each class into probability
    model.add(Activation('softmax'))

    opt = keras.optimizers.Adam(lr=0.0001)

    model.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model
model=CNN_classification_model()
model.summary()
```

Output:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
activation_1 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 29, 29, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 29, 29, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 29, 29, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 27, 27, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 27, 27, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 10816)             0
_________________________________________________________________
dense (Dense)                (None, 128)               1384576
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 1,451,434
Trainable params: 1,451,434
Non-trainable params: 0
_________________________________________________________________
```

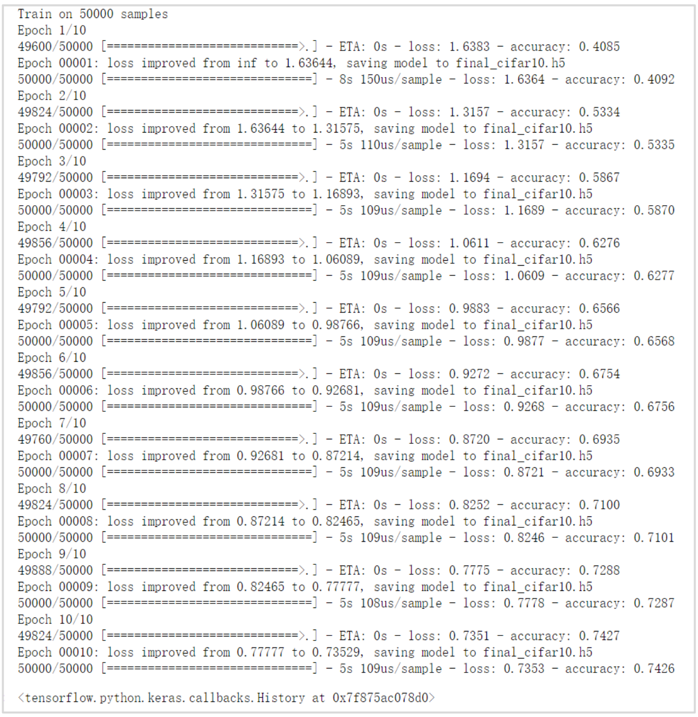
2.3.4 Model Training

Code:

```
from tensorflow.keras.callbacks import ModelCheckpoint
model_name = "final_cifar10.h5"
model_checkpoint = ModelCheckpoint(model_name, monitor='loss',verbose=1, save_best_only=True)

#load pretrained models
trained_weights_path = 'cifar10_weights.h5'
if os.path.exists(trained_weights_path):
    model.load_weights(trained_weights_path, by_name =True)
#train
```

```
model.fit(x_train,y_train, batch_size=32, epochs=10,callbacks = [model_checkpoint],verbose=1)
```

Output:



```
Train on 50000 samples
Epoch 1/10
49600/50000 [===========================>.] - ETA: 0s - loss: 1.6383 - accuracy: 0.4085
Epoch 00001: loss improved from inf to 1.63644, saving model to final_cifar10.h5
50000/50000 [==============================] - 8s 150us/sample - loss: 1.6364 - accuracy: 0.4092
Epoch 2/10
49824/50000 [===========================>.] - ETA: 0s - loss: 1.3157 - accuracy: 0.5334
Epoch 00002: loss improved from 1.63644 to 1.31575, saving model to final_cifar10.h5
50000/50000 [==============================] - 5s 110us/sample - loss: 1.3157 - accuracy: 0.5335
Epoch 3/10
49792/50000 [===========================>.] - ETA: 0s - loss: 1.1694 - accuracy: 0.5867
Epoch 00003: loss improved from 1.31575 to 1.16893, saving model to final_cifar10.h5
50000/50000 [==============================] - 5s 109us/sample - loss: 1.1689 - accuracy: 0.5870
Epoch 4/10
49856/50000 [===========================>.] - ETA: 0s - loss: 1.0611 - accuracy: 0.6276
Epoch 00004: loss improved from 1.16893 to 1.06089, saving model to final_cifar10.h5
50000/50000 [==============================] - 5s 109us/sample - loss: 1.0609 - accuracy: 0.6277
Epoch 5/10
49792/50000 [===========================>.] - ETA: 0s - loss: 0.9883 - accuracy: 0.6566
Epoch 00005: loss improved from 1.06089 to 0.98766, saving model to final_cifar10.h5
50000/50000 [==============================] - 5s 109us/sample - loss: 0.9877 - accuracy: 0.6568
Epoch 6/10
49856/50000 [===========================>.] - ETA: 0s - loss: 0.9272 - accuracy: 0.6754
Epoch 00006: loss improved from 0.98766 to 0.92681, saving model to final_cifar10.h5
50000/50000 [==============================] - 5s 109us/sample - loss: 0.9268 - accuracy: 0.6756
Epoch 7/10
49760/50000 [===========================>.] - ETA: 0s - loss: 0.8720 - accuracy: 0.6935
Epoch 00007: loss improved from 0.92681 to 0.87214, saving model to final_cifar10.h5
50000/50000 [==============================] - 5s 109us/sample - loss: 0.8721 - accuracy: 0.6933
Epoch 8/10
49824/50000 [===========================>.] - ETA: 0s - loss: 0.8252 - accuracy: 0.7100
Epoch 00008: loss improved from 0.87214 to 0.82465, saving model to final_cifar10.h5
50000/50000 [==============================] - 5s 109us/sample - loss: 0.8246 - accuracy: 0.7101
Epoch 9/10
49888/50000 [===========================>.] - ETA: 0s - loss: 0.7775 - accuracy: 0.7288
Epoch 00009: loss improved from 0.82465 to 0.77777, saving model to final_cifar10.h5
50000/50000 [==============================] - 5s 108us/sample - loss: 0.7778 - accuracy: 0.7287
Epoch 10/10
49824/50000 [===========================>.] - ETA: 0s - loss: 0.7351 - accuracy: 0.7427
Epoch 00010: loss improved from 0.77777 to 0.73529, saving model to final_cifar10.h5
50000/50000 [==============================] - 5s 109us/sample - loss: 0.7353 - accuracy: 0.7426

<tensorflow.python.keras.callbacks.History at 0x7f875ac078d0>
```

This experiment is performed on a laptop. The network in this experiment is simple, consisting of four convolutional layers. To improve the performance of this model, you can increase the number of epochs and the complexity of the model.

2.3.5 Model Evaluation

Code:

```
new_model = CNN_classification_model()
new_model.load_weights('final_cifar10.h5')

model.evaluate(x_test, y_test, verbose=1)
```

Output:

```
10000/10000 [==============================] - 1s 66us/sample - loss: 0.8327 - accuracy:
0.7134
[0.8326702466964722, 0.7134]
```

Predict on a single image.

Code:

```
#output the possibility of each class
new_model.predict(x_test[0:1])
```

Output:

```
array([[2.3494475e-03, 6.9919275e-05, 8.1065837e-03, 7.8556609e-01,
        2.3783690e-03, 1.8864134e-01, 6.8611270e-03, 1.2157968e-03,
        4.3428279e-03, 4.6843957e-04]], dtype=float32)
```

Code:

```
#output the predicted label
new_model.predict_classes(x_test[0:1])
```

Output:

```
array([3])
```

Plot the first 4 images in the test set and their corresponding predicted labels.

Code:

```
#label list
pred_list = []

plt.figure()
for i in range(0,4):
    plt.subplot(2,2,i+1)
    #plot
    plt.imshow(x_test[i])
    #predict
    pred = new_model.predict_classes(x_test[0:10])
    pred_list.append(pred)
    #Display actual and predicted labels of images
    plt.title("pred:"+category_dict[pred[i]]+"     actual:"+ category_dict[y_test[i][0]])
    plt.axis('off')
plt.show()
```

Output:

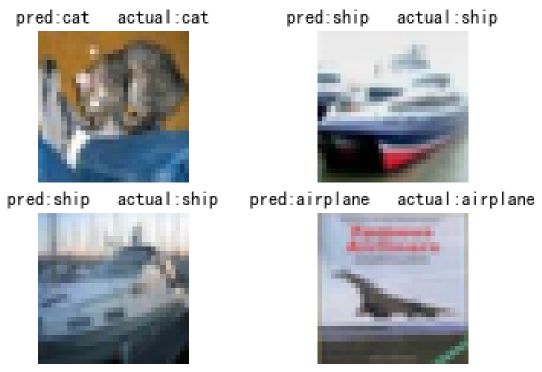


Figure 2-8 The First 4 Images with Predicted Labels

2.4 Summary

This chapter describes how to build an image classification model based on ModelArts.