HCIA-AI

Lab Environment Setup Guide

Issue: 3.0

Huawei Technologies Co., Ltd.

About This Document

Overview

This document is designed for the HCIA-AI certification training course and is intended for people who wish to understand AI and TensorFlow programming basics. You can configure the lab environment based on your actual environment.

Description

This document covers the following four parts:

- Windows lab environment setup
- MacOS lab environment setup
- Ubuntu lab environment setup
- HUAWEI CLOUD user guide

Background Knowledge Required

This course is a basic course for Huawei certification. To better understand this course, familiarize yourself with the following requirements:

- Basic Python knowledge
- Basic concepts of TensorFlow
- Basic Python programming knowledge

Contents

1 Windows Lab Environment Setup

1.1 Installing Anaconda

1.1.1 About Anaconda

Anaconda is a distribution of the Python programming language for scientific computing. It supports Linux, MacOS, and Windows operating systems (OSs) and provides the package and environment management functions. It can easily cope with coexistence and switchover of multiple Python versions and installation of various third-party packages. Anaconda uses the conda tool or command to manage packages and environments. (You can also use pip.) In addition, Anaconda contains Python and related tools. Anaconda is a Python tool for enterprise-level big data analysis. It contains more than 720 open-source packages related to data science and covers data visualization, machine learning, and deep learning. It can be used not only for data analysis, but also in the big data and artificial intelligence (AI) fields.

Anaconda has the following features:

1. After the Anaconda is installed, you do not need to install Python. The Python version is automatically selected during download.

2. During development, you may need different frameworks. In that case, you only need to add virtual environments to Anaconda to achieve development in different environments without compatibility issues. You can configure corresponding environments for special projects to facilitate management.

1.1.2 Procedure

Step 1 Download Anaconda.

Download the installation package at https://www.anaconda.com/distribution/#download-section.

You can select the Windows, Mac, or Linux version based on your OS. In this example, select 64-Bit Graphical Installer under Python 3.8, Windows. (This course does not support a 32-bit computer.)

Figure 1-1

Step 2 Install Anaconda.

Double-click the downloaded **Anaconda3-x.x.x-Windows-x86_64.exe** file. In the displayed dialog box as follows, click **Next**.

Figure 1-2

Click **I Agree**.

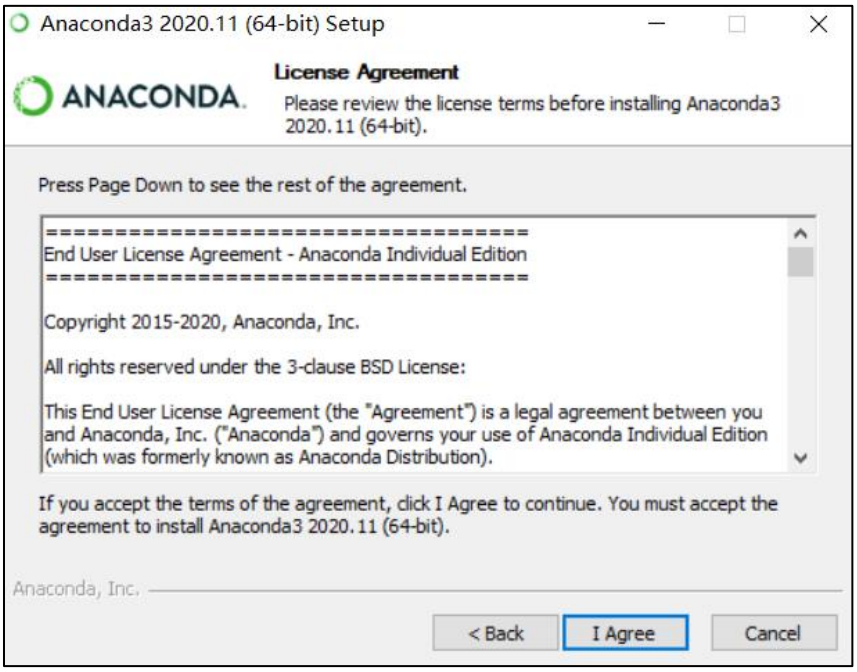


Figure 1-3

Choose **Just me** for **Install for** and click **Next**.

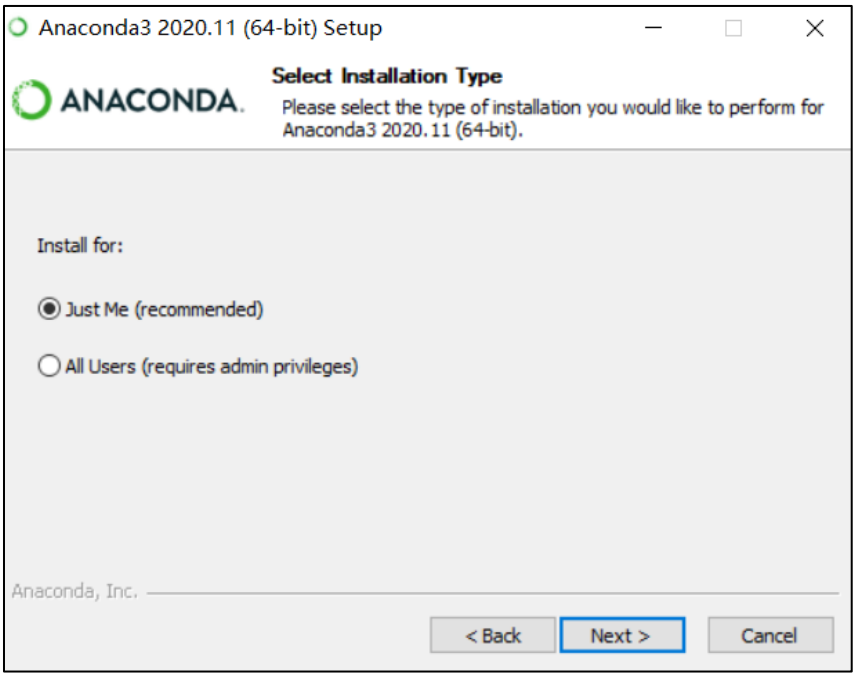


Figure 1-4

Step 3 Choose the installation directory.

Choose the installation directory for the software and click **Next**.

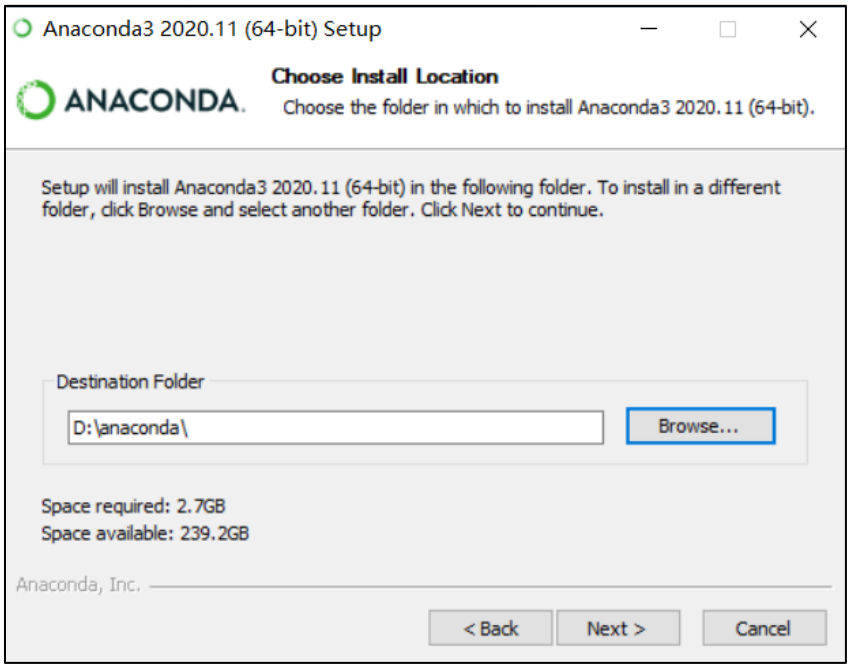


Figure 1-5

Step 4 Set the environment variables.

Select both options. The first option adds environment variables. The second option uses Python 3.8 by default, which reduces subsequent configuration steps. Then, click **Install** to start the installation. (Note: If you have installed another version of Python on the local host, we recommend that you delete it before installing Anaconda. If it is not deleted, deselect the two options. Otherwise, a path error may occur.)

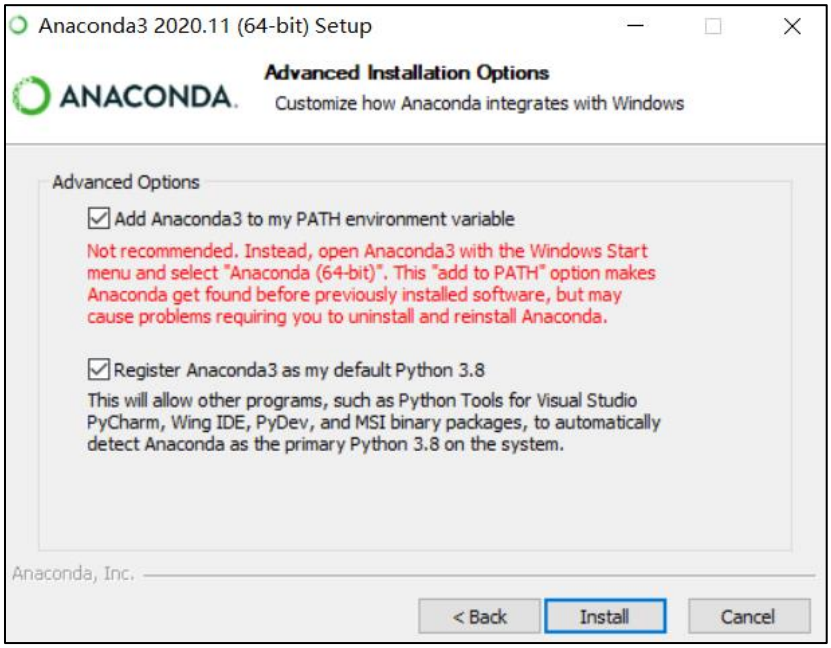


Figure 1-6

After the installation is complete, click **Finish**.

1.2 Changing Channels

1.2.1 Changing the conda Channel

Generally, the use of conda or pip commands downloads required module packages from a server outside China. This takes a long time when the network is poor. Therefore, it is recommended that you download the module from a source in China, which greatly improves the download speed.

On the command prompt interface, run the following command to change the conda channel to the Tsinghua source.

```
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/Anaconda/pkgs/free/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/Anaconda/pkgs/main/
```

See the following example:

```
C:\Users\WWX697589>conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
```

Figure 1-7

Note: When installing Jupyter Notebook or Spyder, it is recommended that you use conda installation. After conda is installed, the corresponding icon is displayed in the start menu, and the virtual environment to which the IDE belongs is marked in the parentheses next to the IDE name to facilitate startup. See the following example:

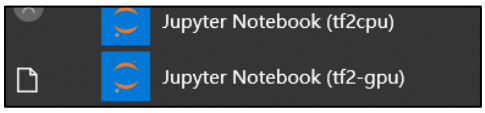


Figure 1-8

1.3 Installing TensorFlow

If you want to install multiple frameworks, create a virtual environment first. For details, see section 1.5 "Anaconda Virtual Environments". To install only one framework, you can directly start the installation after changing the channel.

1.3.1 About TensorFlow

TensorFlow is the second-generation AI learning system developed by Google based on DistBelief. Its name derives from its own operating principle. The graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. TensorFlow is a system that transmits complex data structures to AI neural networks for analysis and processing.

TensorFlow 2.1.0 focuses on simplicity and ease of use, featuring easy to use, powerful, and scalable. It provides the following updates:

- Introduction of the Keras module
- Eager execution (dynamic graph mechanism)
- Support for more platforms and languages
- Removal of deprecated APIs and reduction of duplicate APIs
- Compatibility and continuity

1.3.2 Installing TensorFlow 2.1.0 CPU

Step 1 Install TensorFlow 2.1.0 CPU.

After the Anaconda is installed, open the Terminal window of the Anaconda in the start menu. See the following figure:

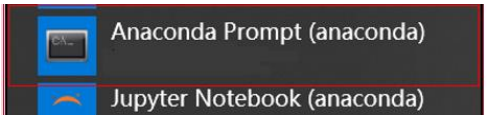


Figure 1-9

Before installation, ensure that the computer is properly connected to the network.

Step 2 Create a virtual environment. A virtual environment is created to ensure that a single framework is available in a single environment, which facilitates development. The relationship between a virtual environment and a framework is similar to that between a virtual machine and a system. When you create a new virtual environment, you can install a new framework. Run the following command to create a virtual environment:

```
conda create –n tf21 python==3.6.5
```

The italic part in red is the name of the virtual environment, which can be customized.

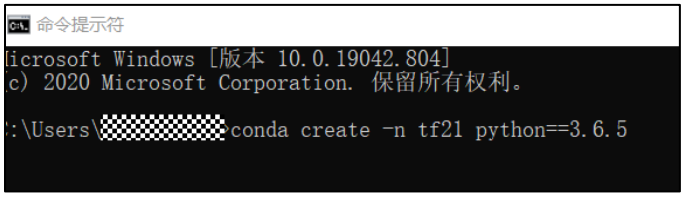


Figure 1-10

Note: TensorFlow 2.1.0 is not compatible with Python 3.7 and later versions. Therefore, it is recommended that you install Python 3.6.5 in the virtual environment.

Step 3 Enter **y** if **y/n** is displayed during the installation.

Step 4 After the installation is complete, run the **activate *tf21*** command to start the virtual environment. The italic part in red is the name of the virtual environment created in Step 2.

Step 5 Run the **pip install tensorflow-cpu==2.1.0** command, as shown in the following figure:

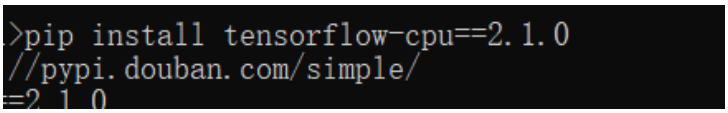


Figure 1-11

If the default channel is not changed, we recommend that you run the following command to change the channel to one in China:

```
pip install tensorflow-cpu==2.1.0 -i https://pypi.douban.com/simple/ --trusted-host pypi.douban.com
```

If **Successfully** is displayed, the installation is successful.

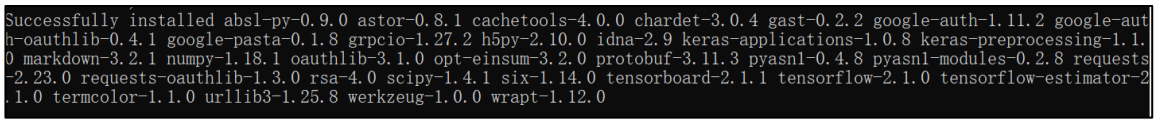


Figure 1-12

Step 6 Python modules required for this exercise, such as pandas numpy scikit-image, are installed along with Anaconda. Therefore, you do not need to install the modules separately. However, if a required module does not exist, run the **pip install + i*nstallation package name*** command to install it. In the follow example, enter **pip install matplotlib**:

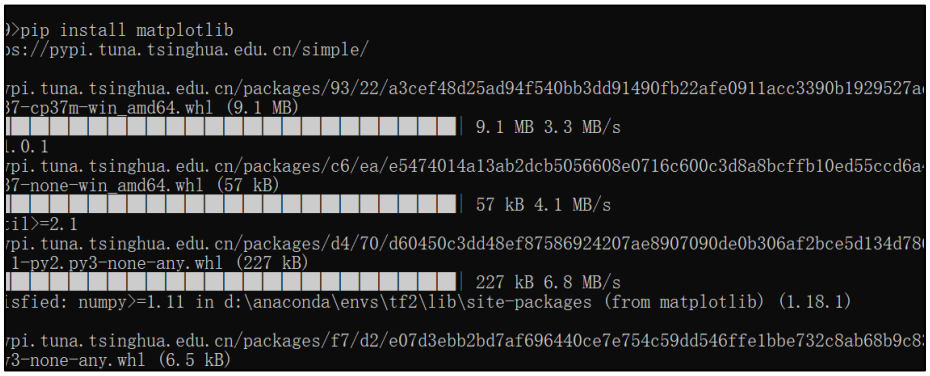


Figure 1-13

Step 7 Check the installation.

Run the **pip list** command to check all installed Python modules. The command output is as follows:

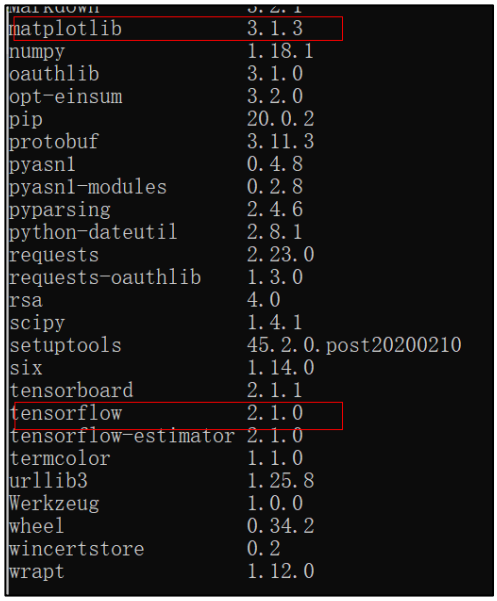


Figure 1-14

1.4 Installing MindSpore

1.4.1 About MindSpore

MindSpore is a Huawei-developed AI computing framework that implements on-demand device-edge-cloud synergy in all scenarios. It provides unified APIs for all scenarios and provides end-to-end capabilities for AI model development, running, and deployment in all scenarios.

MindSpore uses the device-edge-cloud collaborative distributed architecture, new paradigm of differential native programming, and new execution mode of AI Native to achieve better resource efficiency, security, and reliability, lower the AI development threshold in the industry, and release the computing power of Ascend chips, helping achieve inclusive AI.

1.4.2 Installing MindSpore 1.2

Step 1 Open a new command line window and create a virtual environment. For details, see section 1.5 "Anaconda Virtual Environments." Ensure that the Python version is 3.7.5. You can run the following command to specify the version:

```
conda create –n MindSpore python==3.7.5
```

The italic part in red is the environment name, which can be customized.

Note: Python versions other than 3.7.5 are not supported.

Step 2 Run the **activate** command to activate the virtual environment. (The italic part in red is the environment name.)

```
activate MindSpore
```

Step 3 Run the following command to install MindSpore 1.2. Alternatively, go to the official website https://www.mindspore.cn/install to obtain the latest version.

```
pip install https://ms-release.obs.cn-north-4.myhuaweicloud.com/1.2.0/MindSpore/cpu/windows_x64/mindspore-1.2.0-cp37-cp37m-win_amd64.whl --trusted-host ms-release.obs.cn-north-4.myhuaweicloud.com -i https://pypi.tuna.tsinghua.edu.cn/simple
```

Step 4 After the installation succeeds, enter **Python** in the command line window to access the development environment. Run the following command to import MindSpore. If no error is reported, the installation is successful.

```
Import mindspore
```

1.5 Anaconda Virtual Environments

1.5.1 Overview

Anaconda allows you to virtualize multiple Python environments that are irrelevant to each other on the local host. This feature helps boost its popularity. When chaotic dependence exists among different Python modules or different development frameworks exist, you can use virtual environments to isolate them.

Note: Modules are independent of each other in different virtual environments. That is, if you installed TensorFlow in environment A, to use TensorFlow in environment B, you need to install TensorFlow in environment B. This mechanism also applies to Spyder and Jupyter Notebook.

1.5.2 Creating a Virtual Environment Using the Command Prompt

Step 1 Search for cmd in the search box on the taskbar to open the command prompt.

Step 2 Run the **conda create –n *environment name* python==x.x** command. The environment name can be customized. Specify the Python version based on your needs.

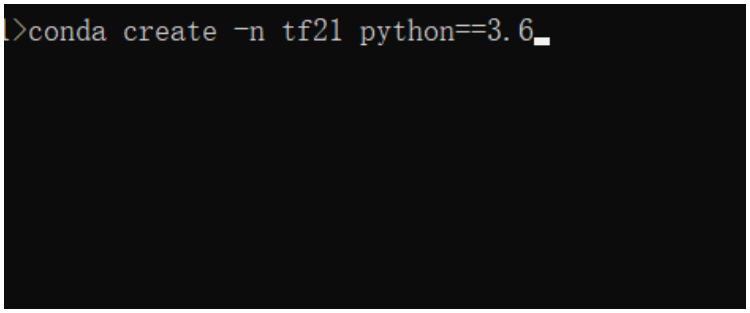


Figure 1-15

Step 3 When the system prompts you to select a value, enter **y**.

Wait until the installation is complete.

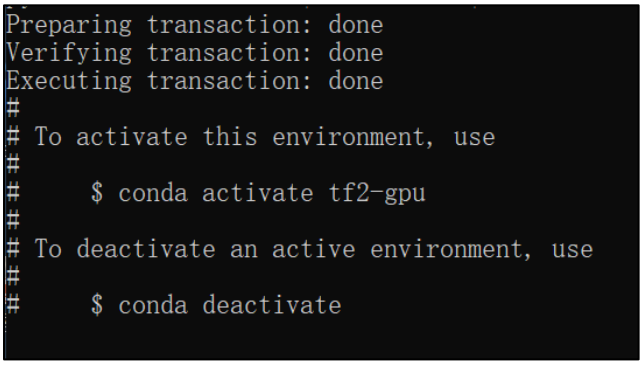


Figure 1-16

1.5.3 Activating a Virtual Environment

In the command prompt, enter the **activate** ***Environment name*** command to activate the corresponding virtual environment. If the name in the square brackets before the command line changes, it indicates that the system has started and entered the environment.

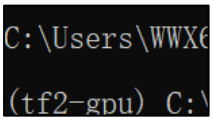


Figure 1-17

Then, you can run pip or conda commands to install the modules.

1.5.4 Viewing a Virtual Environment

You can use the Anaconda Navigator to view the created virtual environment.

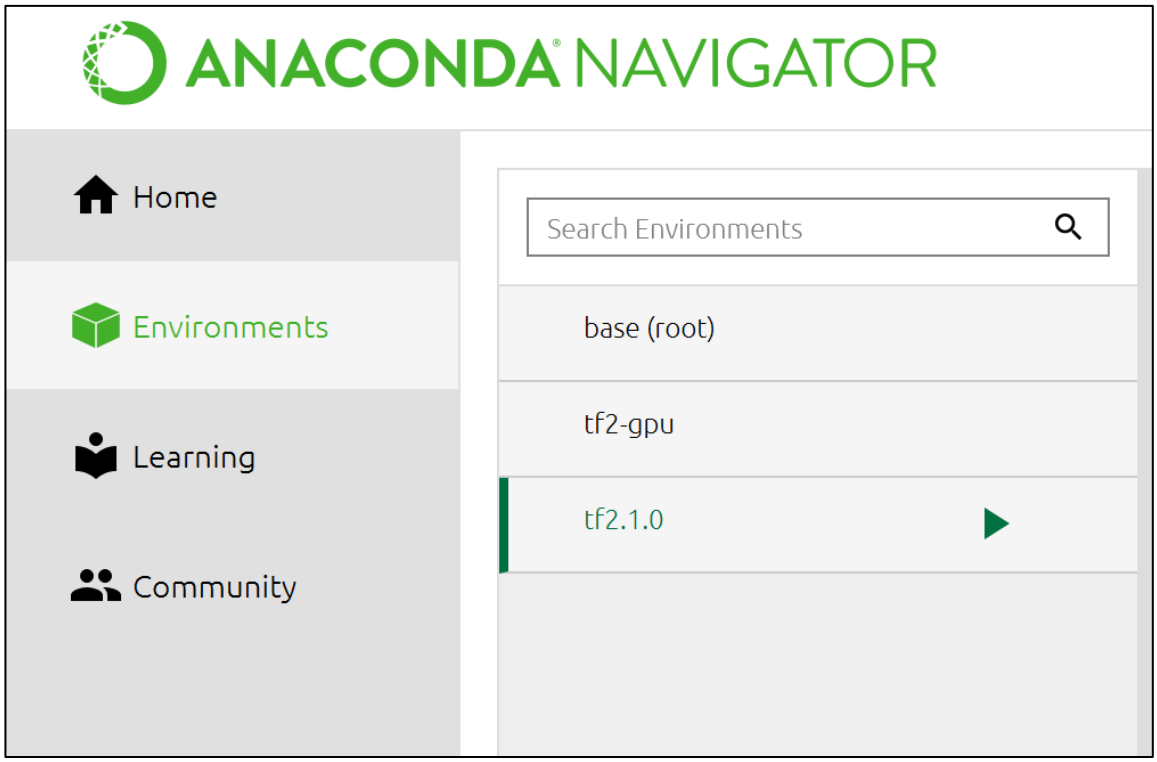


Figure 1-18

You can also click the triangle on the right of the environment name to go to the command prompt window of the virtual environment.

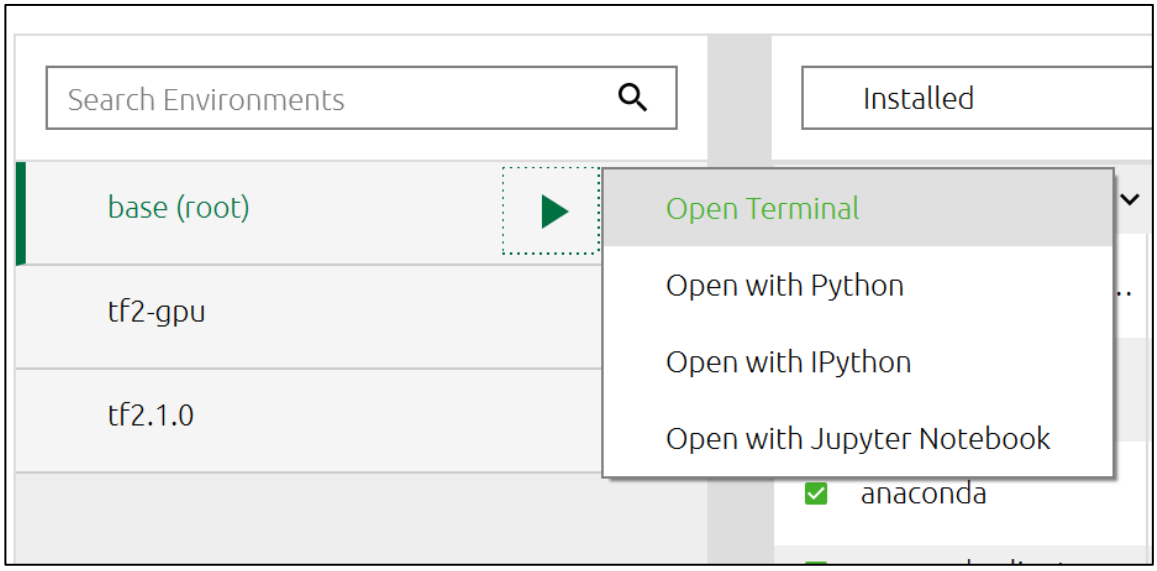


Figure 1-19

In this case, you can install modules in the current environment without starting the virtual environment.

1.5.5 Deleting a Virtual Environment

You can run the **conda remove –n *Environment name* –all** command and enter **y** to delete an environment, as shown in the following figure:

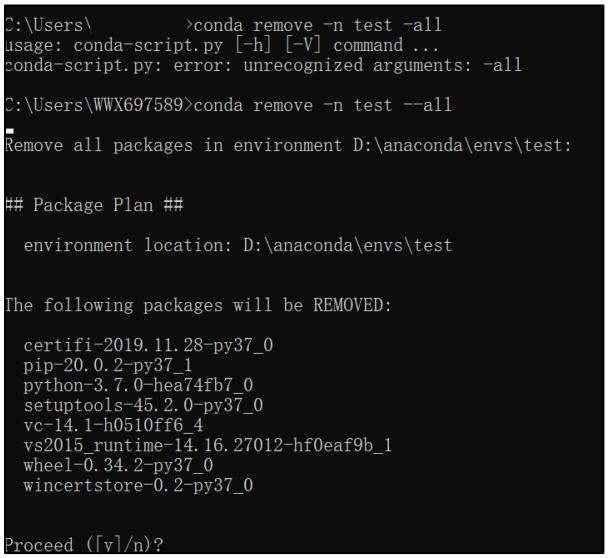


Figure 1-20

1.6 Switching the Kernel in Jupyter Notebook

During development, we usually need to install multiple frameworks in different environments. Therefore, you may need to switch the Python kernel between different environments during the development of a project. This section provides a simple method for switching the kernel.

1.6.1 Checking Jupyter Notebook

The nb_conda_kernels module and Jupyter Notebook must be installed in the same environment. Check whether Jupyter Notebook is installed in the current environment.

Run the following command to view the installation packages in the current environment:

```
conda list
```

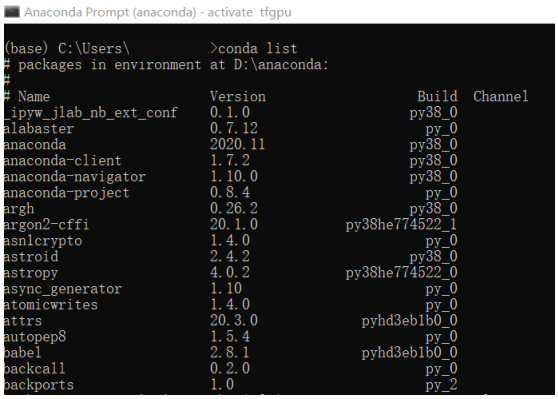


Figure 1-21

If **jupyter** is in the list, Jupyter Notebook is installed in the environment.

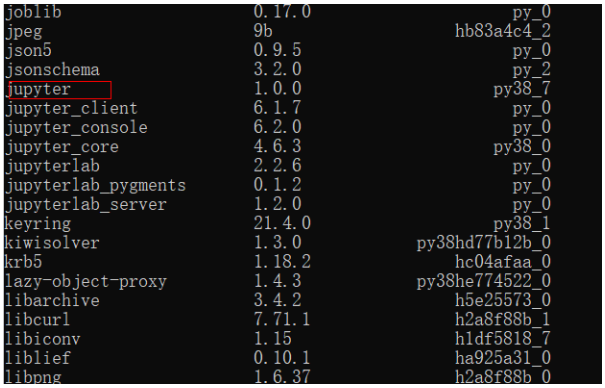


Figure 1-22

Note: If ipykernel is installed in other environments, jupyter_client and jupyter_core are in the list, but not jupyter.

1.6.2 Installing nb_conda_kernels

The nb_conda_kernels module must be installed in the same environment as Jupyter Notebook. Generally, when Anaconda is installed, Jupyter Notebook is installed in the base environment. This environment is the basic environment used when the command prompt window is opened without any activate command.

Run the following command to install nb_conda_kernels in the base environment:

```
conda install nb_conda_kernels
```

Figure 1-23

Note: This module must be installed in the same environment as Jupyter Notebook. On some computers, when the command prompt window is started, (base) is not displayed before the address; however, when the Anaconda prompt window is started, (base) is displayed before the address.

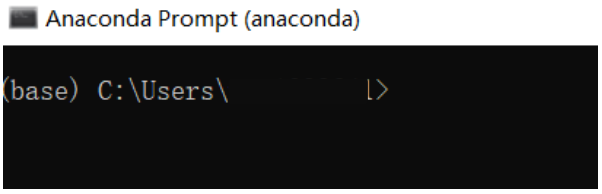


Figure 1-24

1.6.3 Installing ipykernel

After nb_conda_kernels is installed, run the activate command to enter the environment where you want to add the kernel. In this example, the tfgpu environment is used.

Run the following command to enter the environment. (The italic part in red is the environment name, which can be customized.)

```
activate tfgpu
```

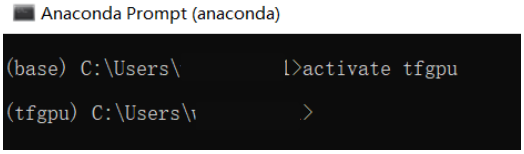


Figure 1-25

Run the **conda install ipykernel** command to install the ipykernel module.

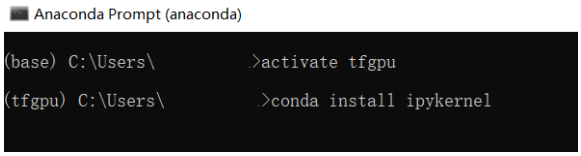


Figure 1-26

After the installation is complete, you can switch to another kernel from the Notebook kernel. Then, if you want to create a virtual environment, you only need to install the ipykernel module in the environment to find the new kernel in Jupyter. You do not need to install nb_conda_kernels again.

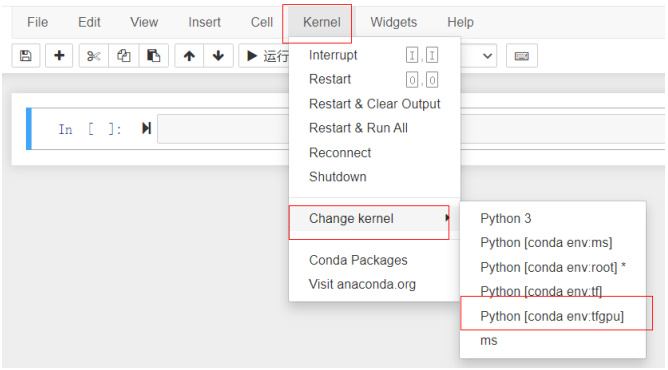


Figure 1-27

2 MacOS Lab Environment Setup

2.1 Overview

2.1.1 About This Lab

This document describes how to set up a development environment for all HCIA-AI exercises based on the MacOS system, including downloading and installing Anaconda, selecting the Python version, installing the dependencies, and installing Jupyter Notebook.

2.1.2 Objectives

Upon completion of this task, you will be able to set up the HCIA-AI development environment of the CPU version based on the MacOS system.

2.1.3 Modules Required for the Exercise

Anaconda 3.7 or later

Python 3.6.5

TensorFlow 2.1.0

2.2 Downloading Anaconda and Configuring the Python Environment

Anaconda is a distribution of the Python programming language for scientific computing. It supports Linux, MacOS, and Windows OS and provides the package and environment management functions. It can easily cope with coexistence and switchover of multiple Python versions and installation of various third-party packages. Anaconda uses the conda tool or command to manage packages and environments. In addition, Anaconda contains Python and related tools. Anaconda is a Python tool for enterprise-level big data analysis. It contains more than 720 open-source packages related to data science and

covers data visualization, machine learning, and deep learning. It can be used not only for data analysis, but also in the big data and AI fields.

After the Anaconda is installed, you do not need to install Python.

2.2.1 Downloading Anaconda

Step 1 Go to the Anaconda official website at https://www.Anaconda.com/ and click **Download** to download the MacOS version, as shown in the following figure:

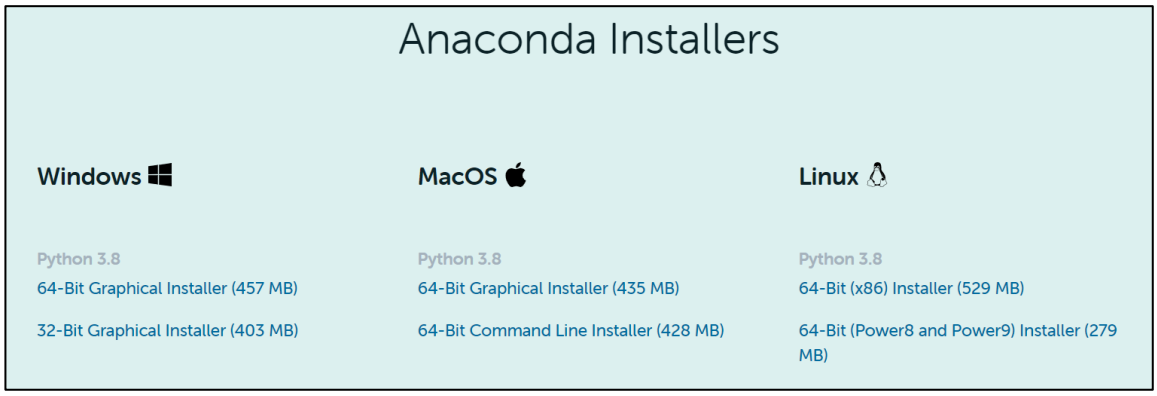


Figure 2-1

Download the Python 3.8 installer (654 MB).

2.2.2 Installing Anaconda

Step 1 Double-click the downloaded Anaconda installation package in .pkg format to install Anaconda.

Step 2 Click **Continue** and then **Install**.

Step 3 Enter the system password to confirm the installation.

Step 4 Wait until the installation is complete.

2.2.3 Creating a Virtual Environment

Step 1 Go to the Anaconda Navigator.

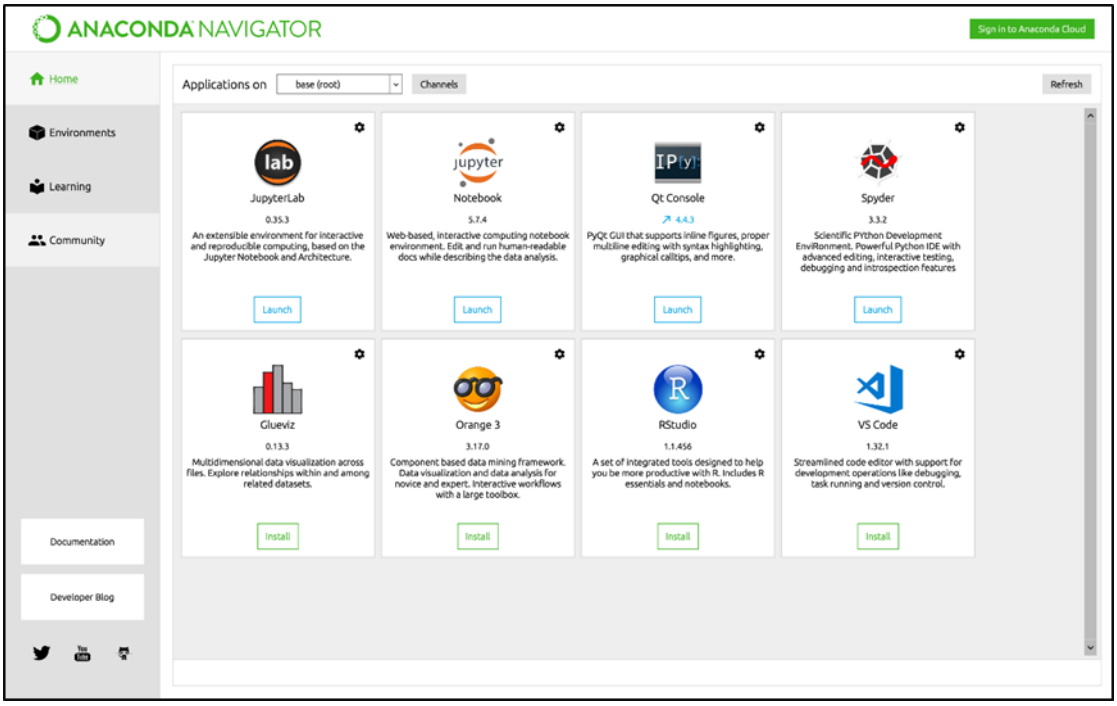


Figure 2-2

Step 2 Click **Environments** in the left navigation pane, and click **Create** in the lower left corner to create a Python development environment of a new version.

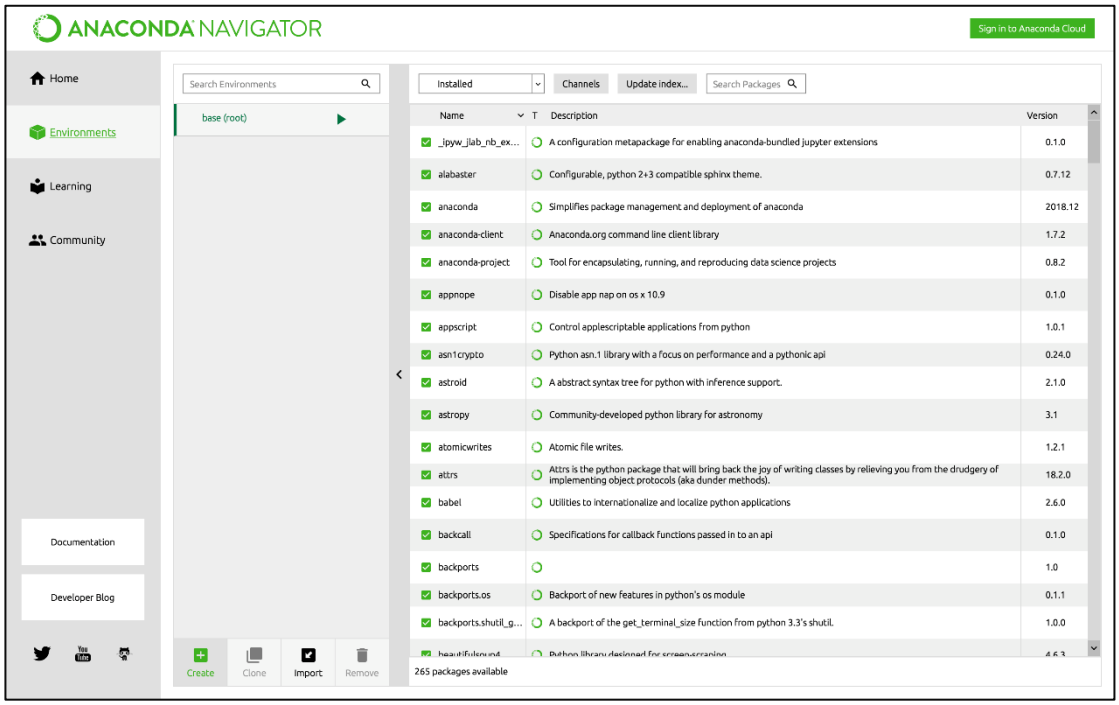


Figure 2-3

Step 3 Set the name as required. Select the Python version. Version 3.6 is recommended.

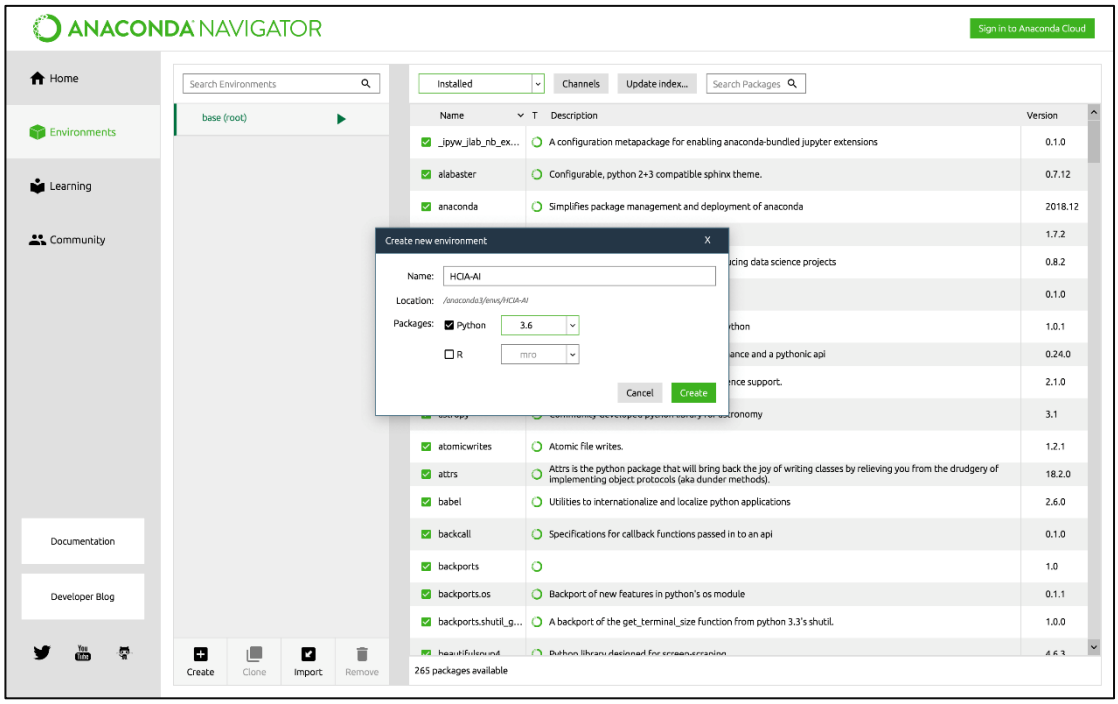


Figure 2-4

Wait until the environment is successfully created.

2.2.4 Testing the Environment

Step 1 Click the triangle icon on the right of the environment name to go to the newly created environment, and choose Open Terminal.

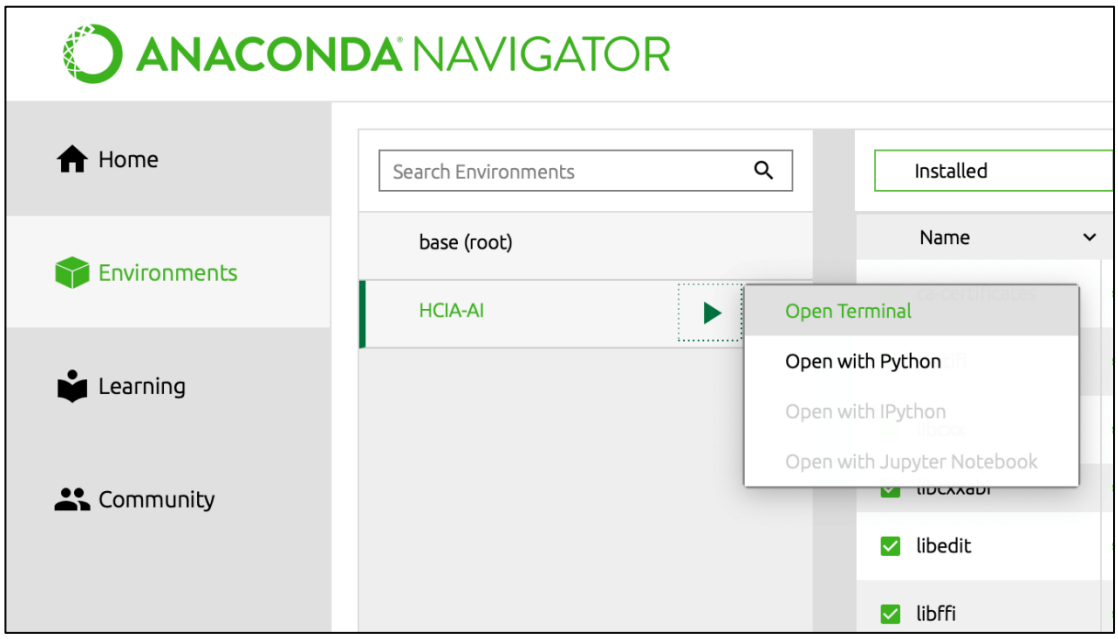


Figure 2-5

Step 2 Enter **python** to verify the Python version of the environment.

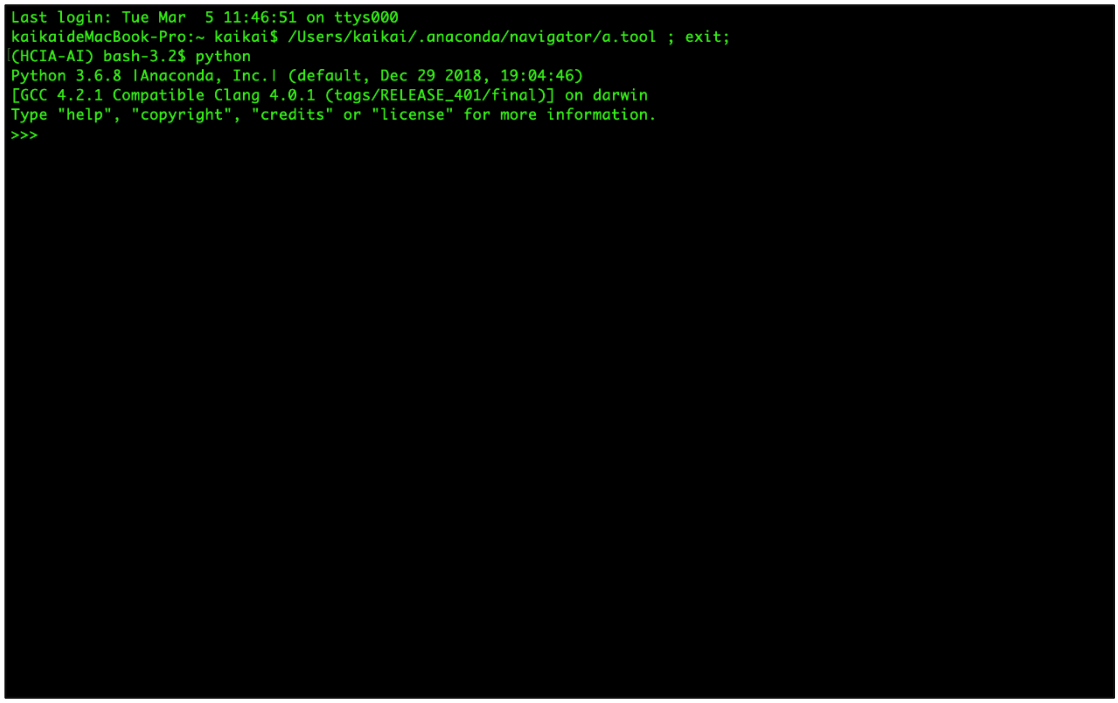


Figure 2-6

If 3.6.8 is displayed, the installation is successful.

2.3 Installing TensorFlow

2.3.1 Installing TensorFlow 2.1.0

Step 1 Create a virtual environment. A virtual environment is created to ensure that a single framework is available in a single environment, which facilitates development. The relationship between a virtual environment and a framework is similar to that between a virtual machine and a system. When you create a new virtual environment, you can install a new framework. Run the following command to create a virtual environment:

```
conda create –n tf21 python==3.6.5
```

The italic part in red is the name of the virtual environment, which can be customized.

Note: TensorFlow 2.1.0 is not compatible with Python 3.7 and later versions. Therefore, it is recommended that you install Python 3.6.5 in the virtual environment.

Step 2 Enter **y** if **y/n** is displayed during the installation.

Step 3 After the installation is complete, run the **activate *tf21*** command to start the virtual environment. The italic part in red is the name of the virtual environment created in Step 2.

Step 4 Run the **pip install tensorflow-cpu==2.1.0** command to install the TensorFlow2.1.0 framework.

Step 5 If an error in the following example is returned, the connotation package is insufficient. In this case, manually install the corresponding package.

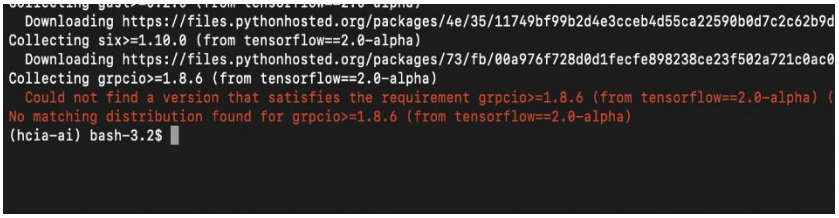


Figure 2-7

Step 6 Run the pip command to install the corresponding packages, and then run the **pip install tensorflow-cpu==2.1.0** command. Then, the TensorFlow is successfully installed.

2.4 Installing Jupyter Notebook

You can install Jupyter Notebook in two ways: using a terminal or using the Anaconda navigator.

2.4.1 Using a Terminal

On the terminal, run the **pip install jupyter notebook** command to install Jupyter Notebook, as shown in the following figure:

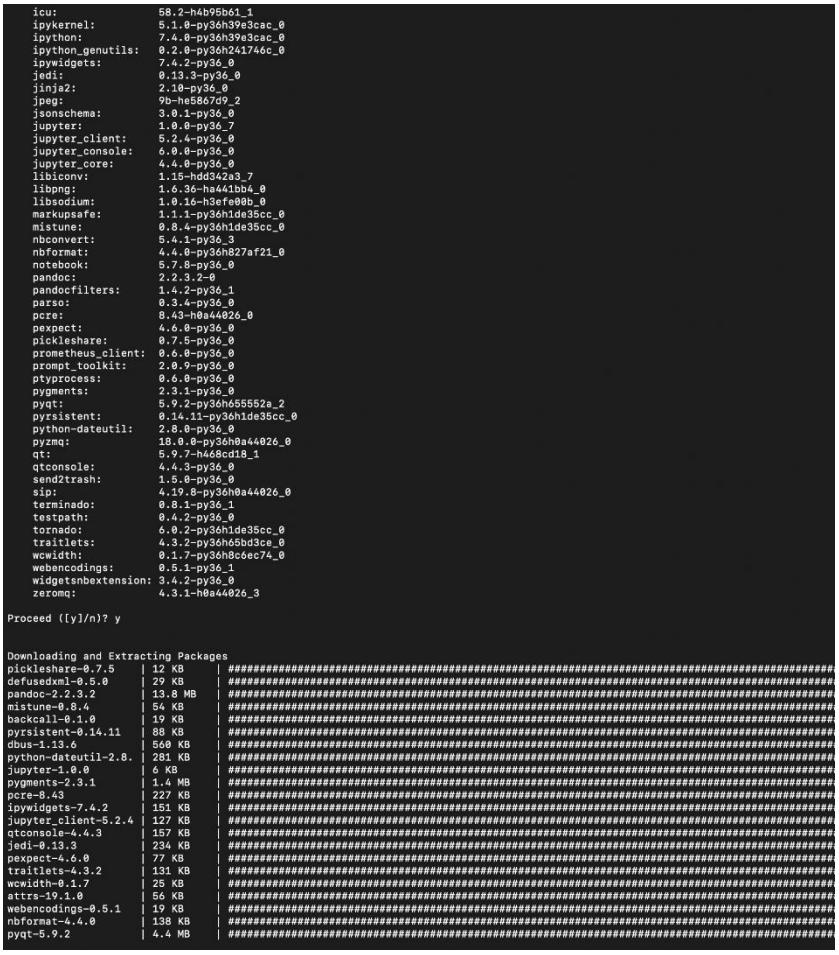

icu: 58.2-h4b95b61_1
ipykernel: 5.1.0-py36h39e3cac_0
ipython: 7.4.0-py36h39e3cac_0
ipython_genutils: 0.2.0-py36h241746c_0
ipywidgets: 7.4.2-py36_0
jedi: 0.13.3-py36_0
jinja2: 2.10-py36_0
jpeg: 9b-he5867d9_2
jsonschema: 3.0.1-py36_0
jupyter: 1.0.0-py36_7
jupyter_client: 5.2.4-py36_0
jupyter_console: 6.0.0-py36_0
jupyter_core: 4.4.0-py36_0
libiconv: 1.15-hdd342a3_7
libpng: 1.6.36-ha441bb4_0
libsodium: 1.0.16-h3efe00b_0
markupsafe: 1.1.1-py36h1de35cc_0
mistune: 0.8.4-py36h1de35cc_0
nbconvert: 5.4.1-py36_3
nbformat: 4.4.0-py36h827af21_0
notebook: 5.7.8-py36_0
pandoc: 2.2.3.2-0
pandocfilters: 1.4.2-py36_1
parso: 0.3.4-py36_0
pcre: 8.43-h0a44026_0
pexpect: 4.6.0-py36_0
pickleshare: 0.7.5-py36_0
prometheus_client: 0.6.0-py36_0
prompt_toolkit: 2.0.9-py36_0
ptyprocess: 0.6.0-py36_0
pygments: 2.3.1-py36_0
pyqt: 5.9.2-py36h655552a_2
pyrsistent: 0.14.11-py36h1de35cc_0
python-dateutil: 2.8.0-py36_0
pyzmq: 18.0.0-py36h0a44026_0
qt: 5.9.7-h468cd18_1
qtconsole: 4.4.3-py36_0
send2trash: 1.5.0-py36_0
sip: 4.19.8-py36h0a44026_0
terminado: 0.8.1-py36_1
testpath: 0.4.2-py36_0
tornado: 6.0.2-py36h1de35cc_0
traitlets: 4.3.2-py36h65bd3ce_0
wcwidth: 0.1.7-py36h8c6ec74_0
webencodings: 0.5.1-py36_1
widgetsnbextension: 3.4.2-py36_0
zeromq: 4.3.1-h0a44026_3
Proceed ([y]/n)? y
Downloading and Extracting Packages
pickleshare-0.7.5 | 12 KB
defusedxml-0.5.0 | 29 KB
pandoc-2.2.3.2 | 13.8 MB
mistune-0.8.4 | 54 KB
backcall-0.1.0 | 19 KB
pyrsistent-0.14.11 | 88 KB
dbus-1.13.6 | 560 KB
python-dateutil-2.8. | 201 KB
jupyter-1.0.0 | 6 KB
pygments-2.3.1 | 1.4 MB
pcre-8.43 | 227 KB
ipywidgets-7.4.2 | 151 KB
jupyter_client-5.2.4 | 127 KB
qtconsole-4.4.3 | 157 KB
jedi-0.13.3 | 234 KB
pexpect-4.6.0 | 77 KB
traitlets-4.3.2 | 131 KB
wcwidth-0.1.7 | 28 KB
attrs-19.1.0 | 56 KB
webencodings-0.5.1 | 19 KB
nbformat-4.4.0 | 138 KB
pyqt-5.9.2 | 4.4 MB

Figure 2-8

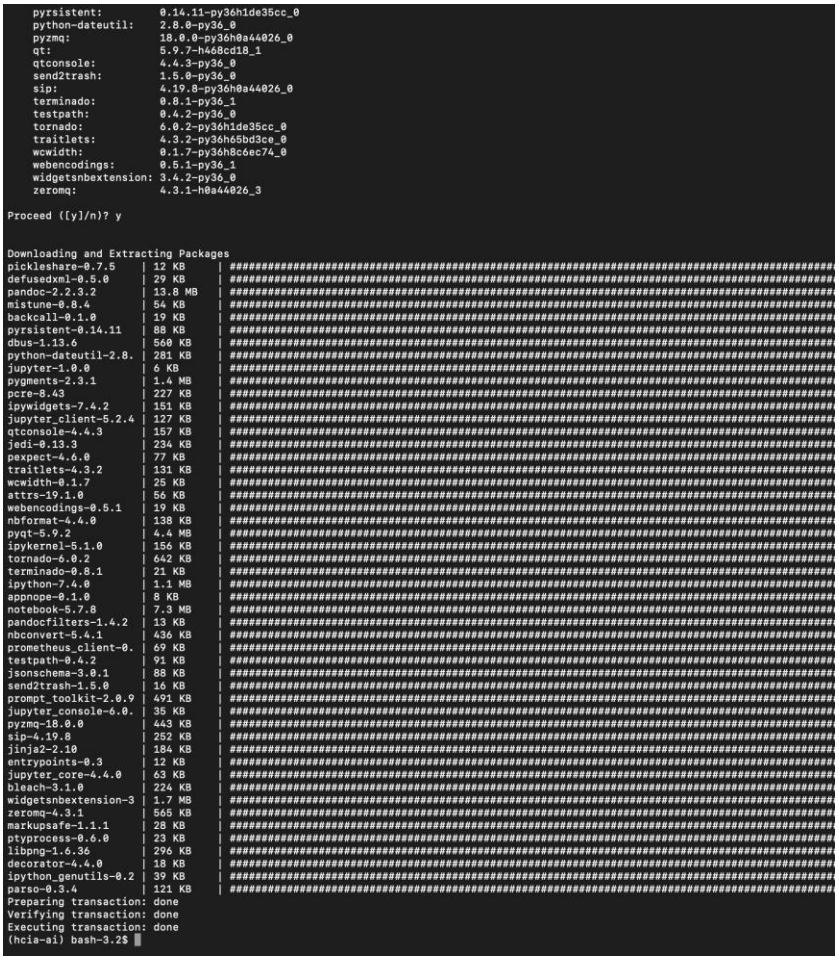


Figure 2-9

The Jupyter Notebook is successfully installed.

2.4.2 Using Anaconda Navigator

Step 1 Go to the Anaconda home page.

Go to the Anaconda home page and click Home on the left.

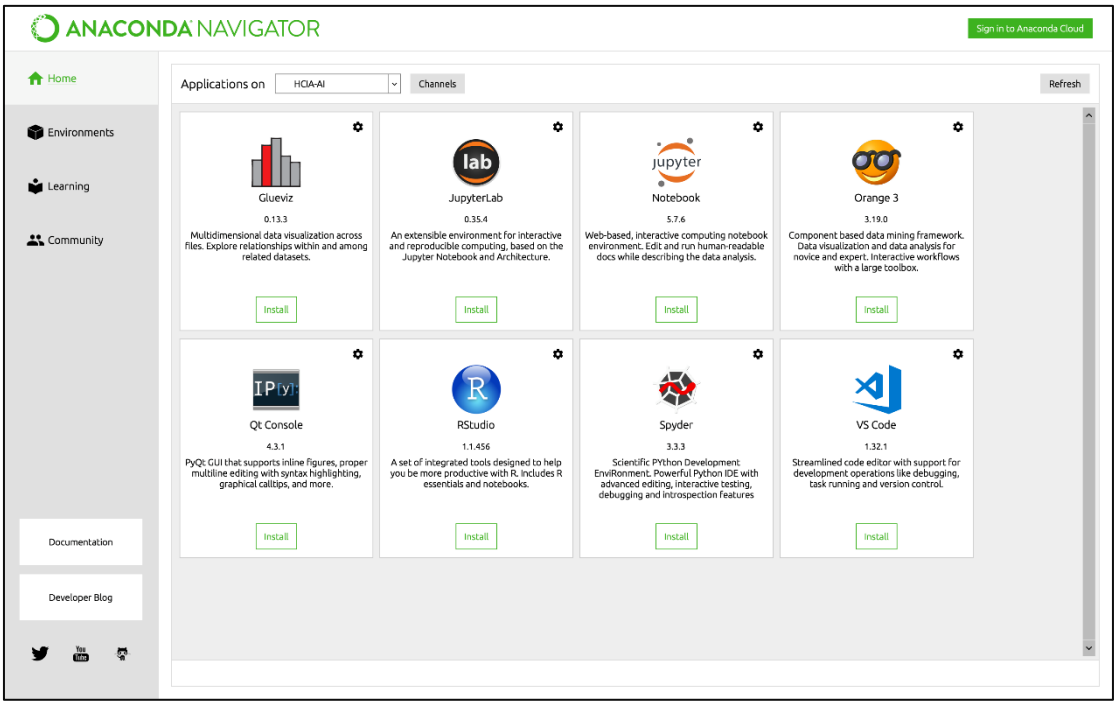


Figure 2-10

Step 2 Install Jupyter Notebook.

In **Application on**, choose **the environment** and click **Install** under **Jupyter Notebook**. After the installation is complete, the following page is displayed:

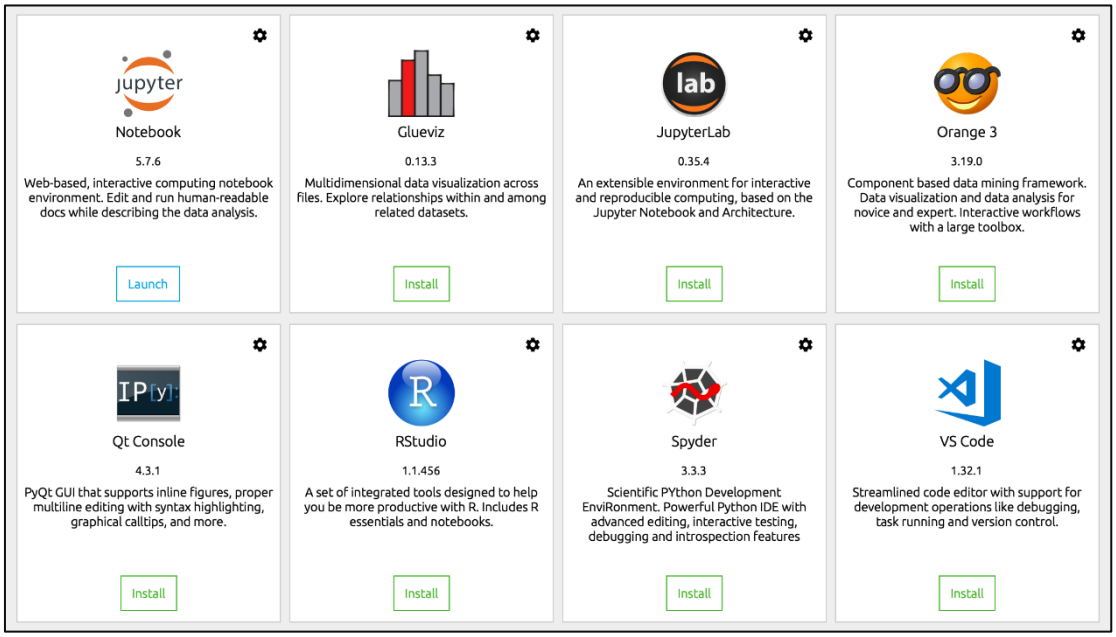


Figure 2-11

The **Install** button under **Jupyter Notebook** changes to **Launch**.

Step 3 Verify the installation result.

Click **Launch** under **Jupyter Notebook**. The Jupyter home page is displayed, as shown in the following figure:

Figure 2-12

Step 4 Click **New** in the upper right corner and choose **python 3** to create a Jupyter file.

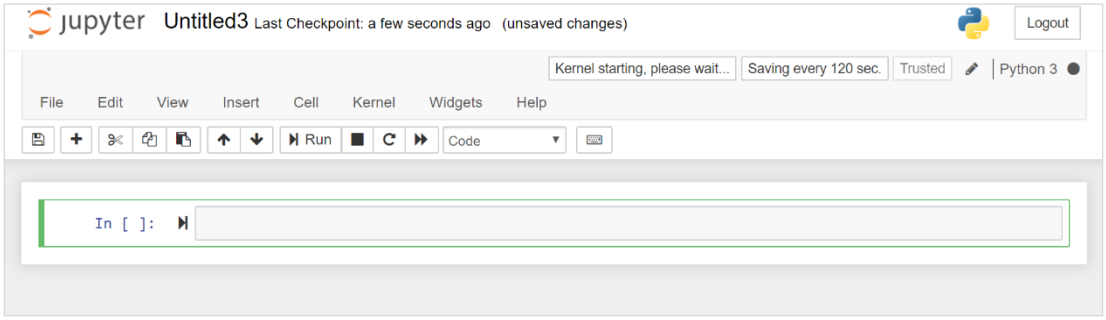


Figure 2-13

Enter the following code in the text box:

```
for i in range(100):
  if i%2 == 0:
    print(i)
```

Click **Run**. The result is as shown in the following figure:

Figure 2-14

2.5 Testing the Environment

This section describes how to verify that the Python 3.6.5 and TensorFlow development environments created upon Anaconda can run properly.

The exercise content is to construct the constant addition operation of the TensorFlow graph mechanism based on the Python 3.6.5, Jupyter, and TensorFlow frameworks to test whether the HCIA-AI environment built based on Anaconda can run properly.

Step 1 Go to the Jupyter page.

Click **Launch** under **Jupyter**. The Jupyter page is displayed, as shown in the following figure:

Figure 2-15

Step 2 Create a Jupyter project.

Click **New** in the upper right corner and choose **python 3** to create a script.

Figure 2-16

Click **Unnamed** in the upper left corner and name the script as **tf_demo**.

Enter the following code in the text box:

```
##Import modules
import tensorflow as tf
import numpy as np

# Create two constant tensors
mat1 = tf.constant(np.array([[1.0, 2.0]]))
mat2 = tf.constant(np.array([[2.],[2.]]))

# Create a multiplication operator
mul = tf.matmul(mat1,mat2)

print(mul)
```

The result is as shown in the following figure:

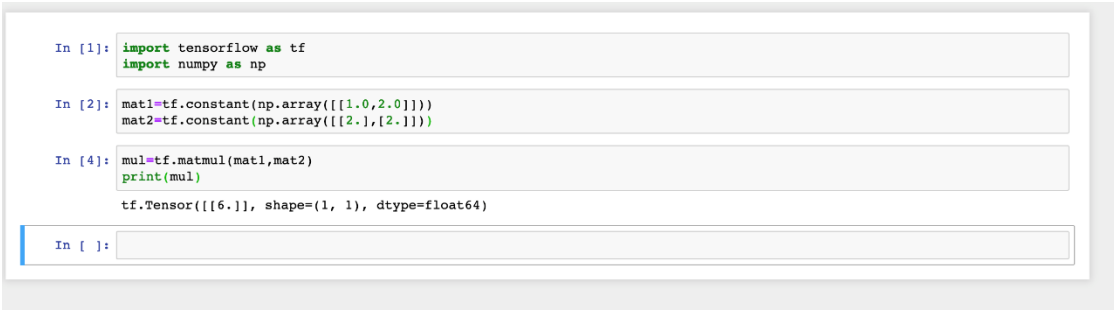


Figure 2-17

Step 3 Test Keras.

The main update of TensorFlow2 is the integration of the Keras module. Therefore, the Keras module is tested here.

Enter the following code:

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss=['sparse_categorical_crossentropy'],
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test)
```

Click **Run** above the code to run the code. The result is as follows:

Figure 2-18

This exercise mainly verifies whether the HCIA-AI lab environment is successfully set up. If no error is reported during code execution, the lab environment is set up successfully.

3 Ubuntu Lab Environment Setup

3.1 Installing Miniconda

Download the 64-bit Miniconda installation package for Linux from https://docs.conda.io/en/latest/miniconda.html. If you are located in China, the download speed at the official website is slow. In this case, you can use the Tsinghua source. Use the installation package with x86_64 in its name.

Miniconda3-py38_4.8.2-Linux-x86_64.sh 85.7 MiB 2020-03-12 00:09

Figure 3-1 Miniconda (Ubuntu) installation package download page

Step 1 Find the downloaded file, right-click the file, choose **Properties** from the shortcut menu, and click **Permissions**. Select **Allow executing files as program** for **Execute** at the bottom.

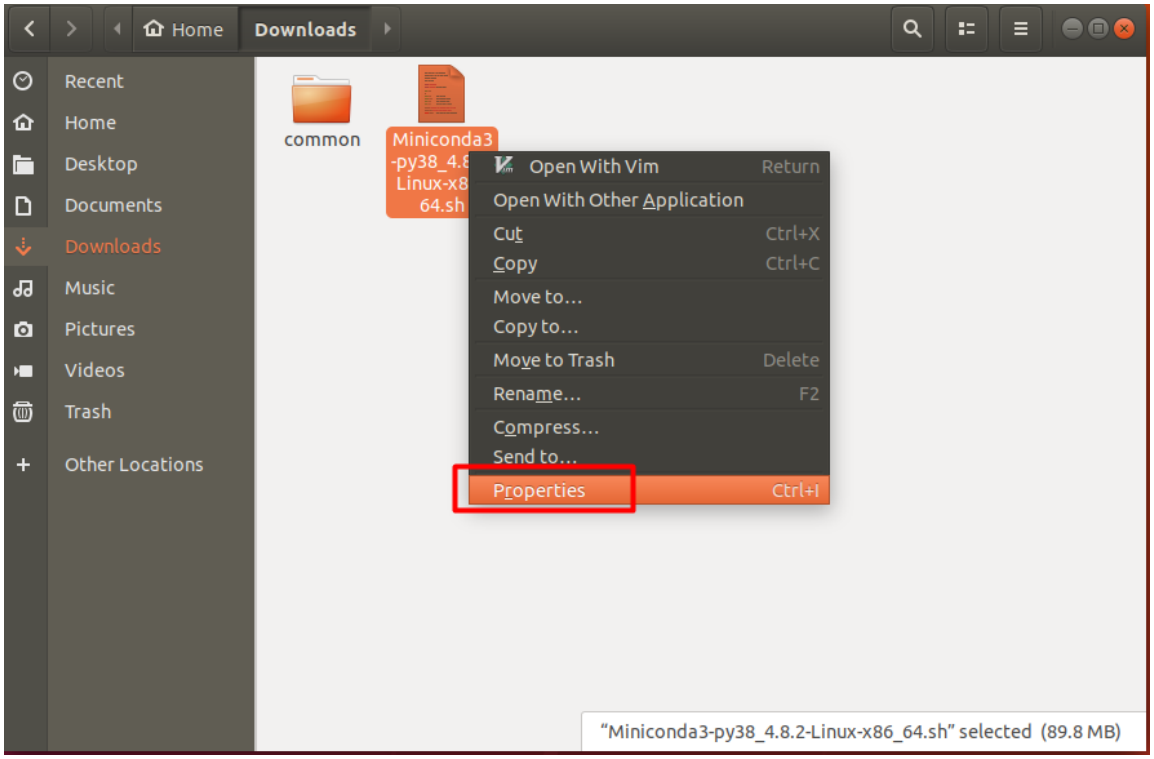


Figure 3-2 Configuring the file execution permission (1)

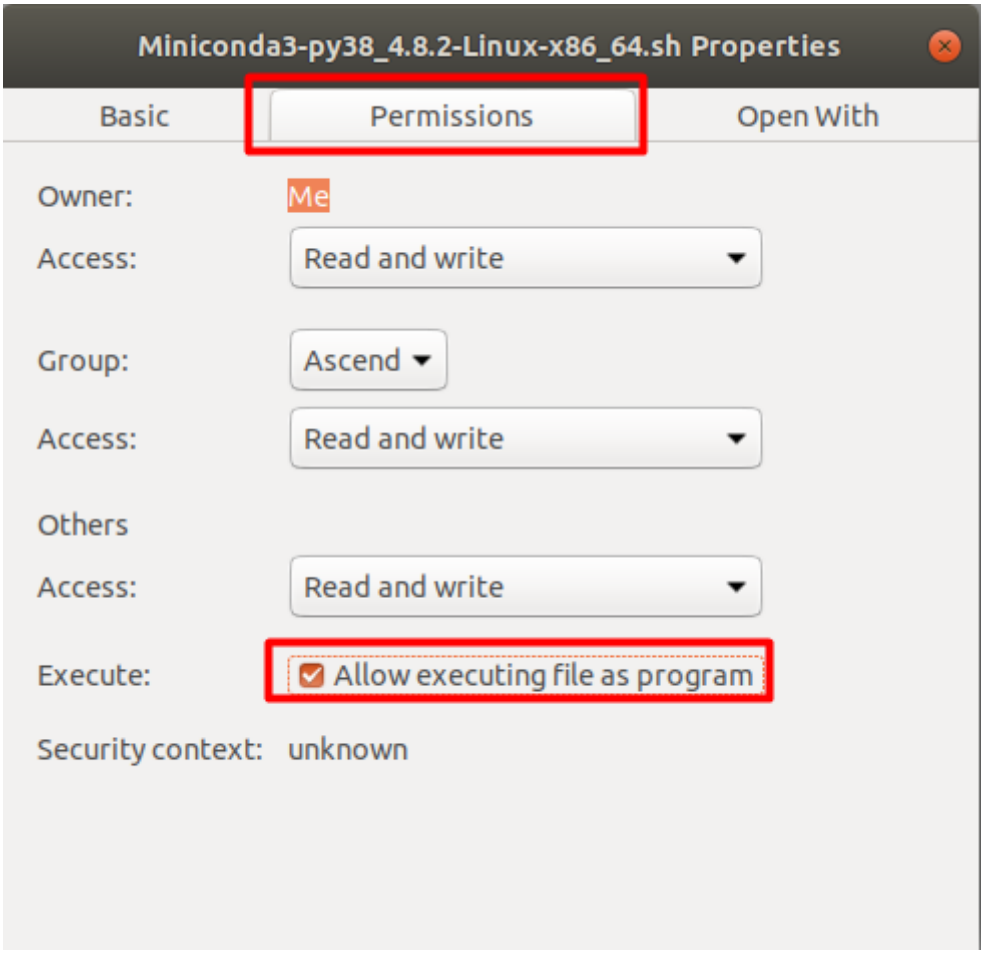


Figure 3-3 Configuring the file execution permission (2)

Step 2 Right-click the blank area in the folder where the file is located and choose **Open in Terminal**. If you log in as a common user, run the following command to switch to the bash mode:

```
bash
```

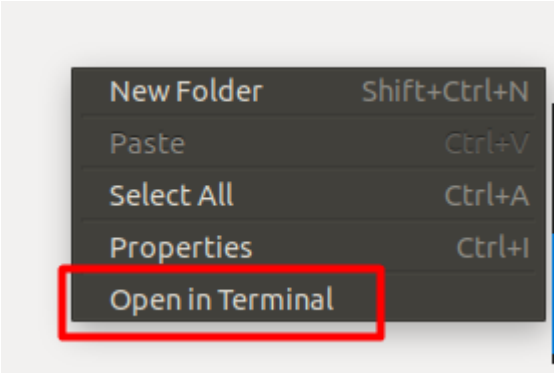


Figure 3-4 Open in Terminal

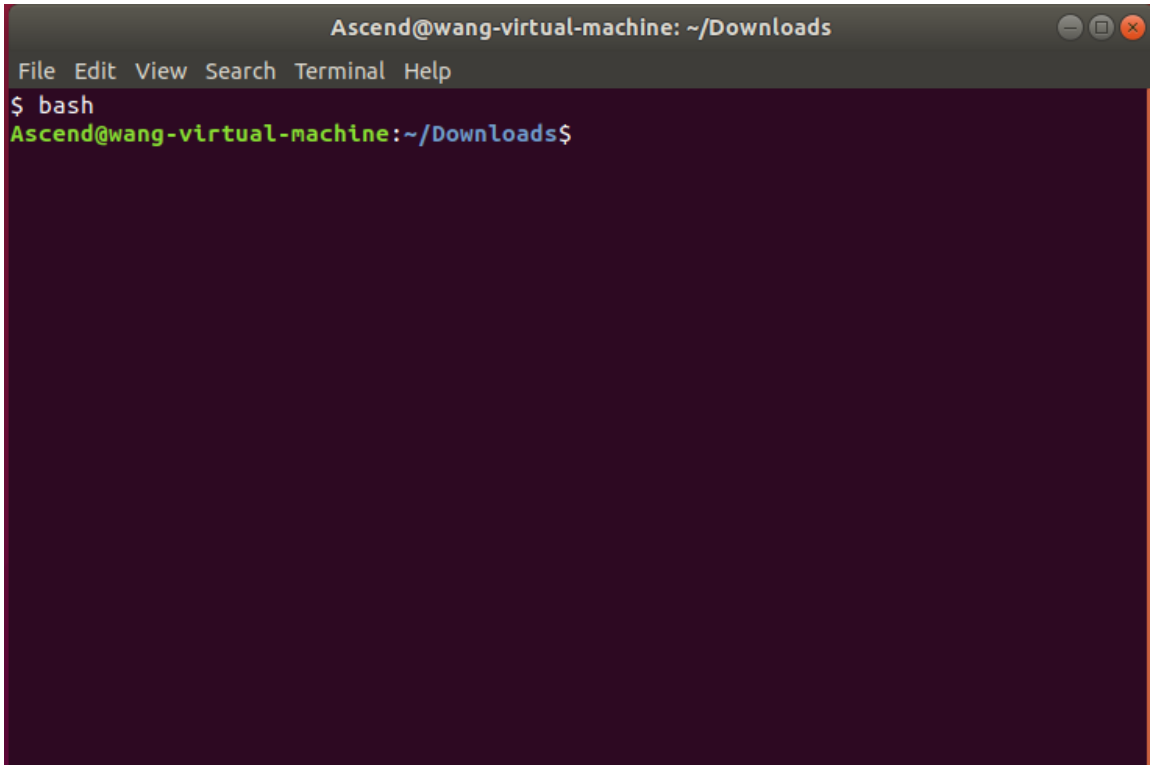


Figure 3-5 Switching to the bash mode

Step 3 Run the following command to execute the installation file. Use the actual name of the downloaded file in the command.

```
./Miniconda3-py38_4.8.2-Linux-x86_64.sh
```

Step 4 During the installation, enter **yes** to agree to the installation terms. The default value is **no**.

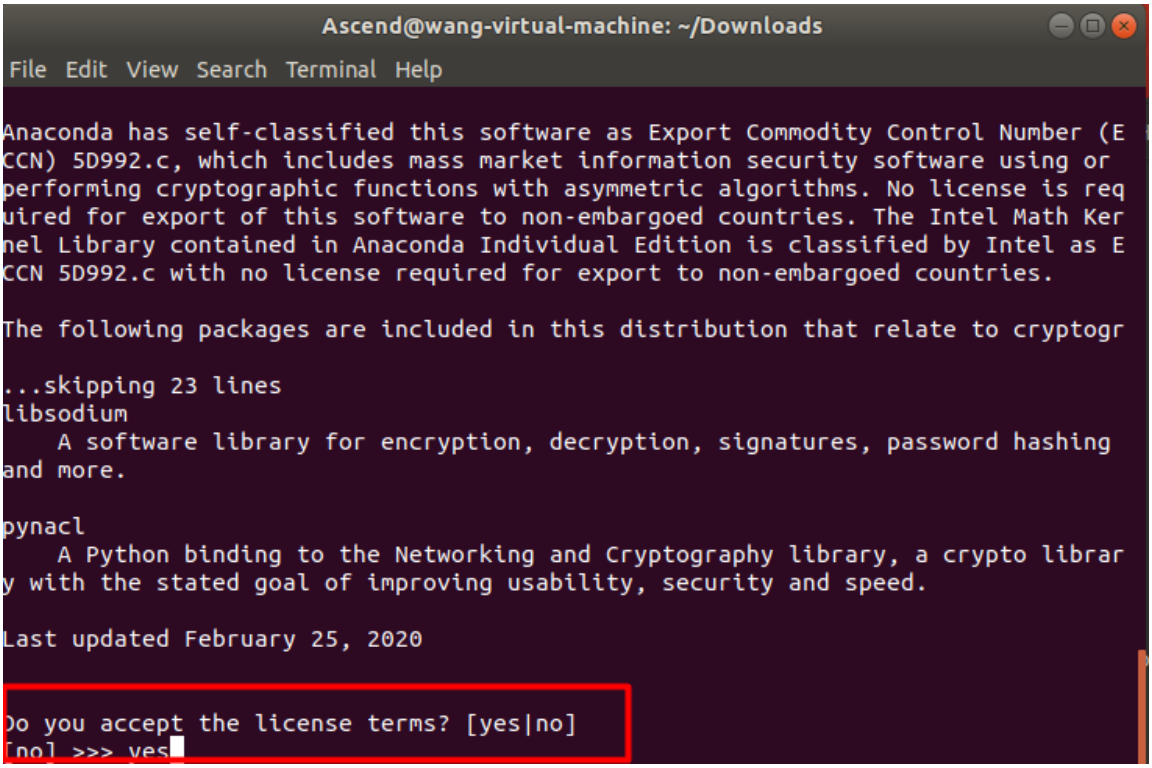


Figure 3-6 Agreeing to the installation terms

Step 5 After the installation is complete, enter **yes** to initialize Miniconda.

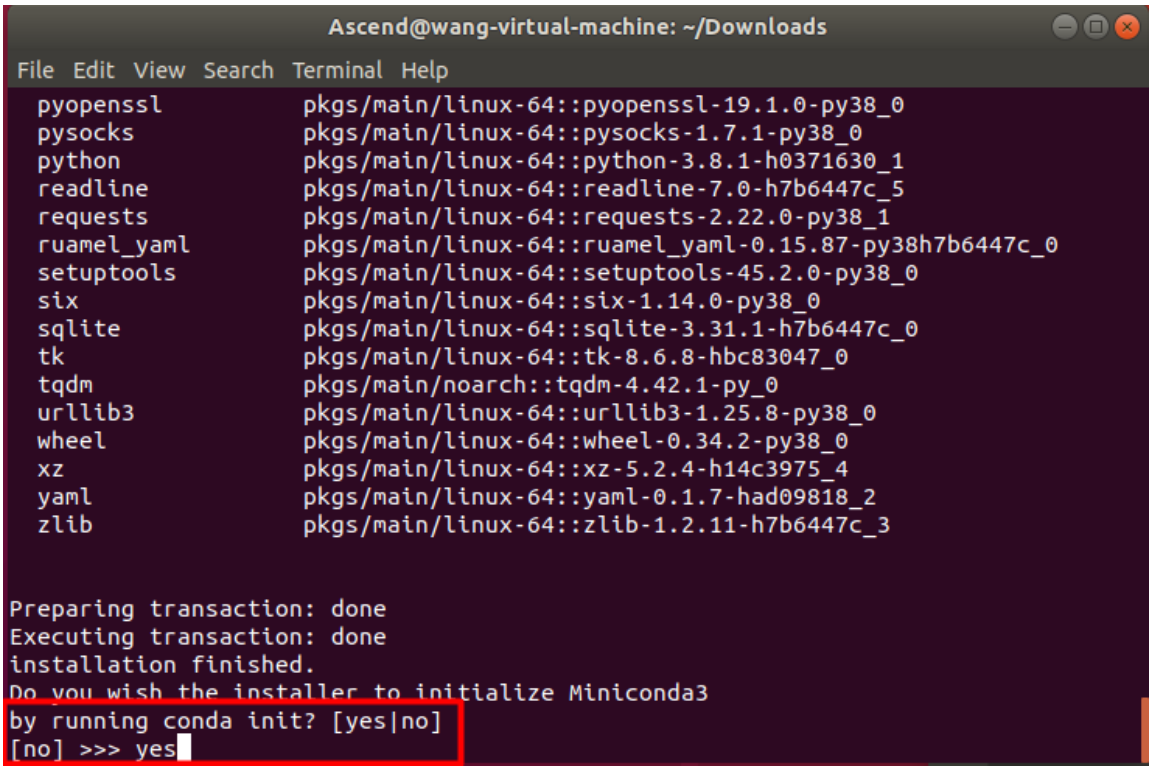


Figure 3-7 Confirming the initialization of Miniconda

Step 6 After the preceding operations are complete, the Miniconda is successfully installed. Close the current terminal and open a new terminal to perform subsequent operations.

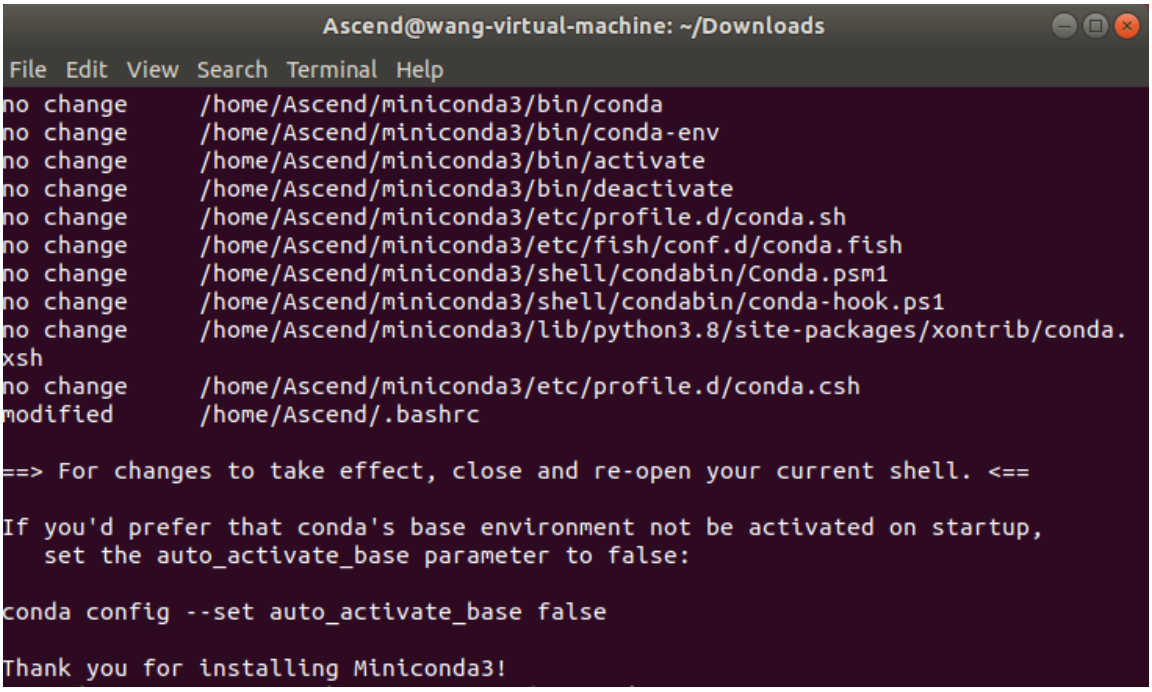


Figure 3-8 Miniconda installed

3.2 Creating Virtual Environments

Step 1 MindSpore and TensorFlow require a large number of dependencies. If they are installed in the same environment, errors may occur. Therefore, you need to create different virtual environments for different frameworks. Open the command line window and run the following commands to create a virtual environment for MindSpore and TensorFlow CPU, respectively. During the process, enter **y** for confirmation.

```
conda create –n MindSpore python==3.7.5
conda create –n TensorFlow-CPU python==3.6.5
```

Step 2 Run the following command to activate the virtual environment:

```
conda activate MindSpore
```

3.3 Changing the pip Channel

You can install Python in pip or conda mode. Because the two modes are not fully compatible with each other, you are advised to install Python in either mode. In this exercise, the pip mode is used.

Step 1 Open a new terminal and run the following commands in sequence to update the index and install Vim:

```
sudo apt-get update
sudo apt-get install vim
```

Step 2 Run the following commands to create and edit the pip configuration file:

```
mkdir ~/.pip/
touch ~/.pip/pip.conf
vim ~/.pip/pip.conf
```

Step 3 Press i to enter the editing mode, copy the following content to the file. Press Esc, enter **:**, and then enter **wq!**. Save the configuration and exit.

```
[global]
index-url = https://mirrors.huaweicloud.com/repository/pypi/simple
trusted-host = mirrors.huaweicloud.com
timeout = 120
```

3.4 Installing MindSpore

Step 1 Open a new command line window and create a virtual environment. For details, see section 1.5 "Anaconda Virtual Environments." Ensure that the Python version is 3.7.5. Run the activate command to activate the virtual environment.

```
activate MindSpore
```

Step 2 Run the following command to install MindSpore 1.2. Alternatively, go to the official website https://www.mindspore.cn/install to obtain the latest version.

```
pip install https://ms-release.obs.cn-north-4.myhuaweicloud.com/1.2.0/MindSpore/cpu/ubuntu_x86/mindspore-1.2.0-cp37-cp37m-linux_x86_64.whl --trusted-host ms-release.obs.cn-north-4.myhuaweicloud.com -i https://pypi.tuna.tsinghua.edu.cn/simple
```

Step 3 After the installation succeeds, enter **Python** in the command line window to access the development environment. Run the following command to import MindSpore. If no error is reported, the installation is successful.

Note: Due to dependency problem of the GLIBC package, Ubuntu 16.04 does not support MindSpore. Use Ubuntu 18 or later.

```
import mindspore
```

3.5 Installing TensorFlow (CPU Version)

Step 1 Open a new command line window. Ensure that the Python version is 3.6.5. Run the **activate** command to activate the virtual environment. (The italic part in red is the name of the virtual environment.)

```
activate TensorFlow-CPU
```

Step 2 Run the following command to install the latest version of TensorFlow. If you need to specify a TensorFlow version, enter another line to specify the version. For details, visit https://tensorflow.google.cn/install.

```
pip install tensorflow-cpu==2.1.0
```

Step 3 After the installation succeeds, enter **Python** in the command line window to access the development environment. Run the following command to import TensorFlow. If no error is reported, the installation is successful.

```
import tensorflow
```

4 HUAWEI CLOUD User Guide

4.1 Overview

HUAWEI CLOUD provides abundant cloud resources and cloud services to meet developers' requirements. This document describes how to apply for HUAWEI CLOUD resources, including account application and real-name authentication, access key (AK) and secret access key (SK) obtaining, and main services of HUAWEI CLOUD.

After reading this manual, you will be able to:

- Understand the specific content of the AI development environment;
- Apply for an AI development environment;
- Perform function tests based on the established environment.

4.2 Account Application and Real-Name Authentication

4.2.1 HUAWEI CLOUD Accounts

You can use a HUAWEI CLOUD account to:

- Enable corresponding services on the HUAWEI CLOUD official website to carry out the exercises in the course.
- Experience various convenient and cost-effective cloud services provided by HUAWEI CLOUD, including deep learning, machine learning, image recognition, speech recognition, and natural language processing.

4.2.2 Registration

Step 1 Go to the HUAWEI CLOUD official website.

Visit the HUAWEI CLOUD official website at http://www.huaweicloud.com/, as shown in the following figure:

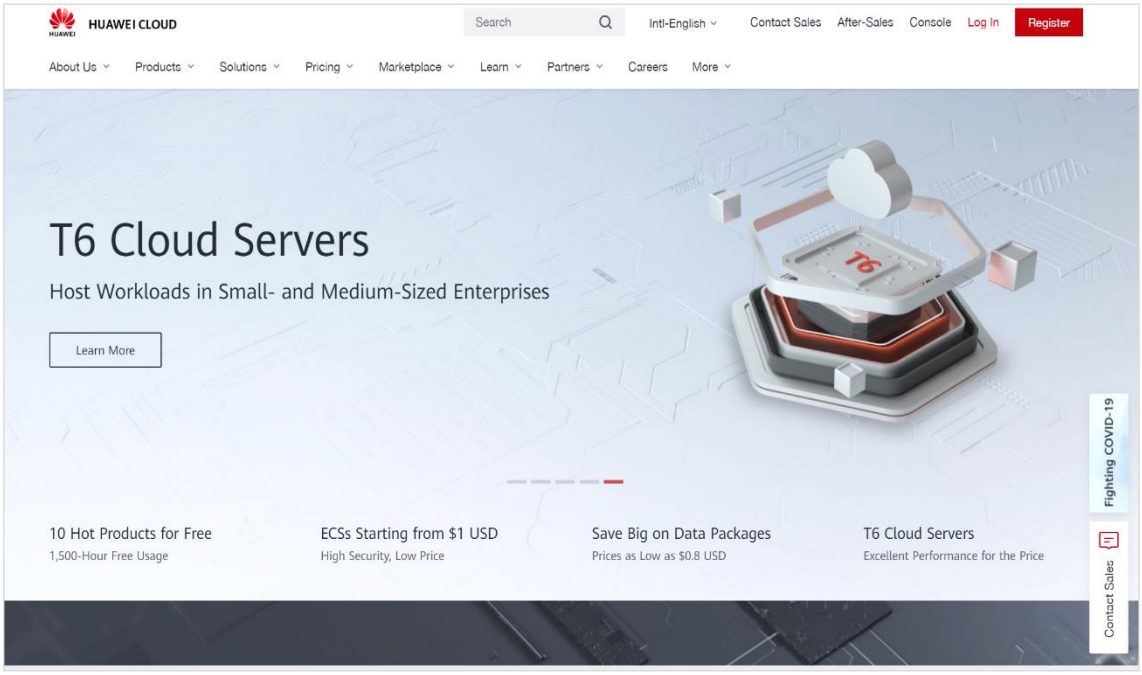


Figure 4-1

Step 2 Click **Register** in the upper right corner of the home page.

Click **Register** in the upper right corner to register an account, as shown in the following figure:

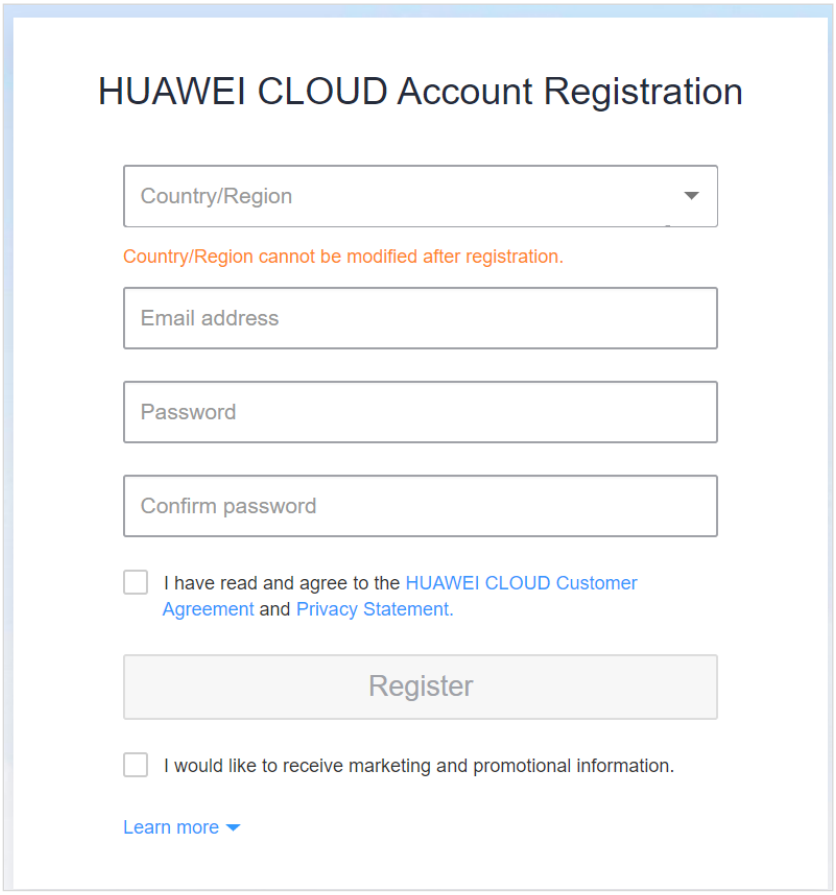


Figure 4-2

Step 3 Verify the real name.

Click **Real-Name Authentication** to perform real-name authentication for the account. Choose **Individual** to go to the next step.

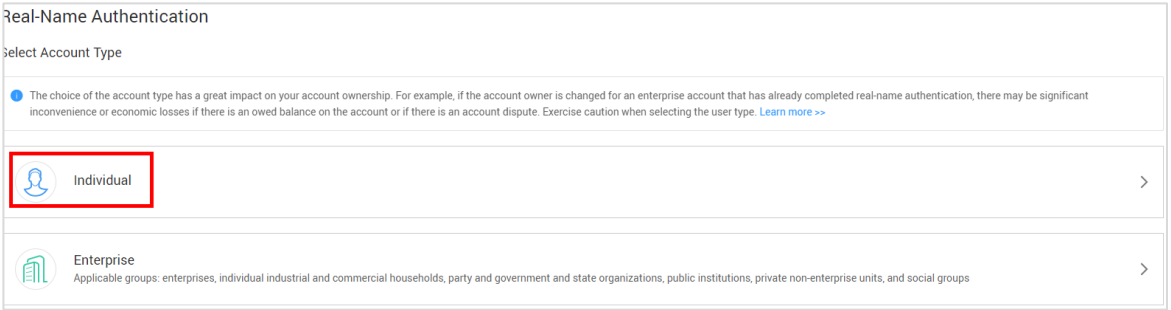


Figure 4-3

Choose **Individual Bank Card Authentication** to go to the next step.

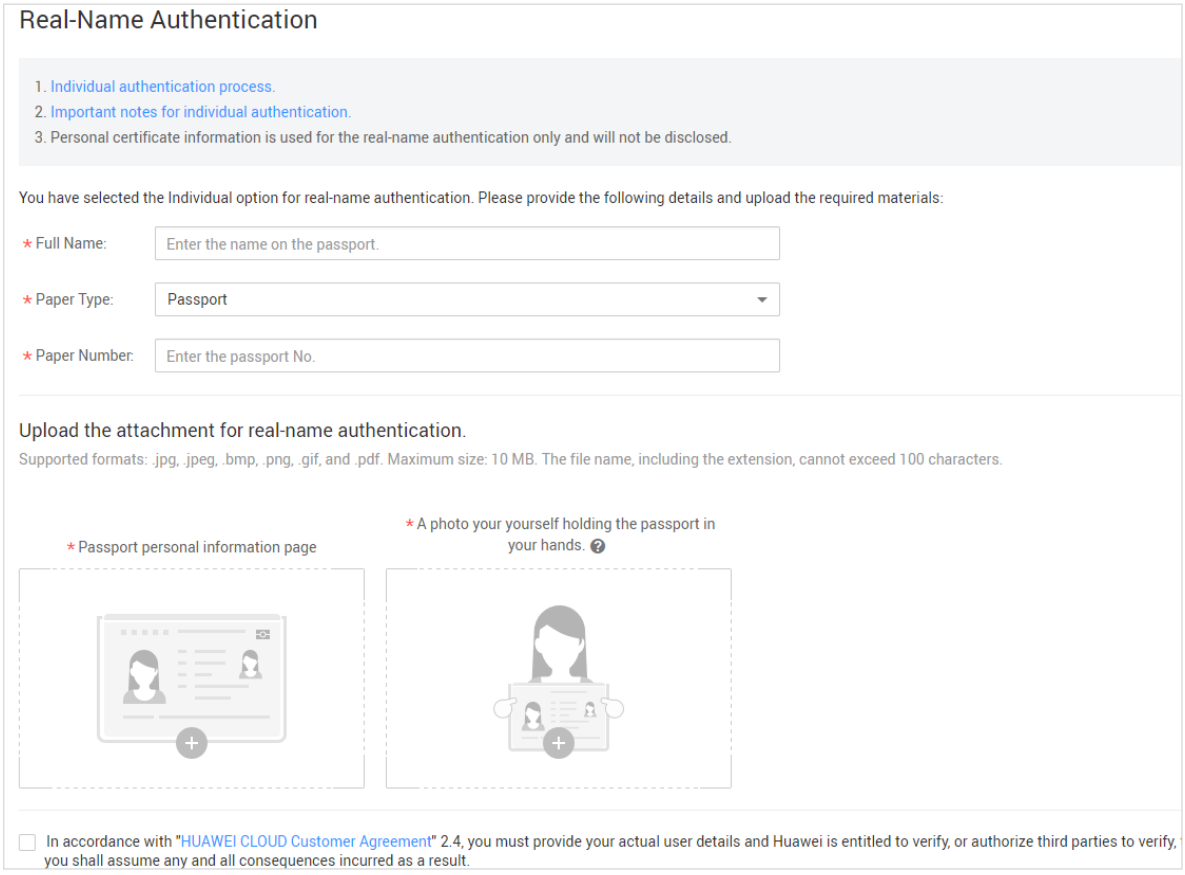


Figure 4-4

Enter the related information and click **Submit**.

After you pass the real-name authentication, you can proceed with other operations.

4.2.3 Application

HUAWEI CLOUD provides various cloud services. You can apply for cloud resources based on your tasks and requirements.

Follow the application wizard of the corresponding service that you apply for.

4.3 Obtaining the AK and SK

4.3.1 Overview

An AK and SK are the identification codes required for the use of HUAWEI CLOUD services. When you use some HUAWEI CLOUD services for the first time, you need to enter the AK and SK, as shown in the following figure:

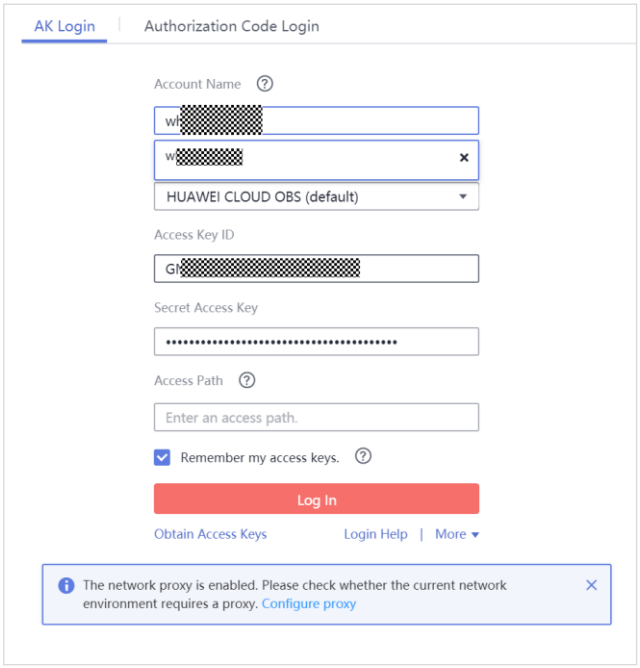


Figure 4-5

In the preceding figure, enter the AK in **Access Key ID** and the SK in **Secret Access Key**. The AK and SK are required when you use the OBS.

4.3.2 Generating the AK and SK

Step 1 Log in to HUAWEI CLOUD. On the home page, go to the account center in the upper right corner, as shown in the following figure:

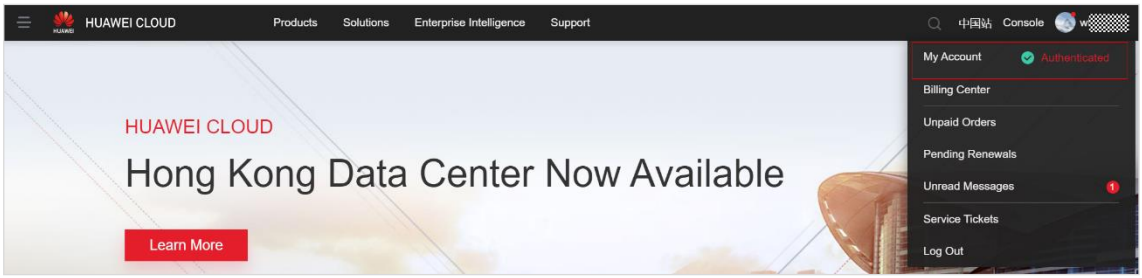


Figure 4-6

Step 2 On the **Account Center** page, click **Manage My Credentials** in the **Security Credentials** row.

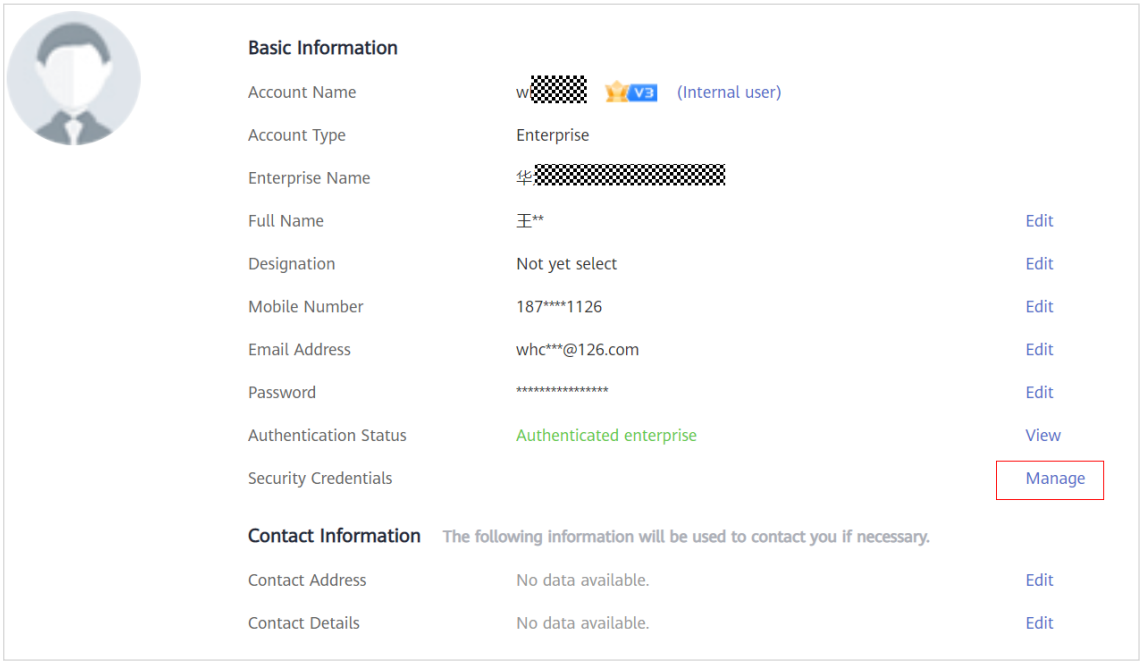


Figure 4-7

Step 3In the list on the left, click **Access Keys**.

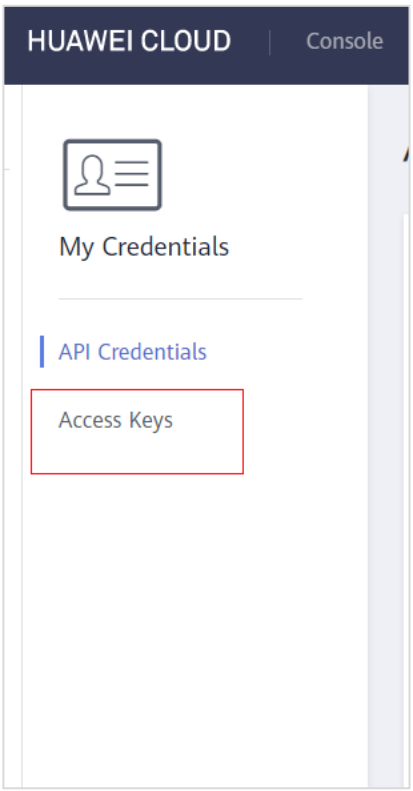


Figure 4-8

Step 4 On the **Access Keys** page, click **New Access Key** to create an AK. You can also edit, disable, or delete an AK by clicking the corresponding button in the **Actions** column.

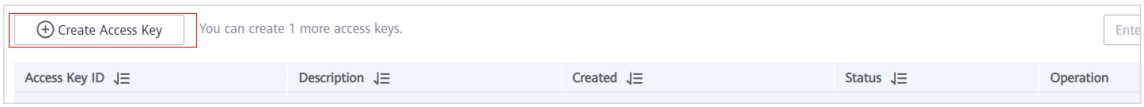


Figure 4-9

Note: A table file is automatically downloaded after you create an AK. The HUAWEI CLOUD AK is stored in the table file. Therefore, keep the file with caution.

4.4 Commonly Used HUAWEI CLOUD Products

ModelArts is a one-stop AI development platform. It provides mass data preprocessing, semi-automated data labeling, distributed training, automated model building, and on-demand deployment of device-edge-cloud models to help AI developers build models quickly and manage the AI development lifecycle during machine learning and deep learning.

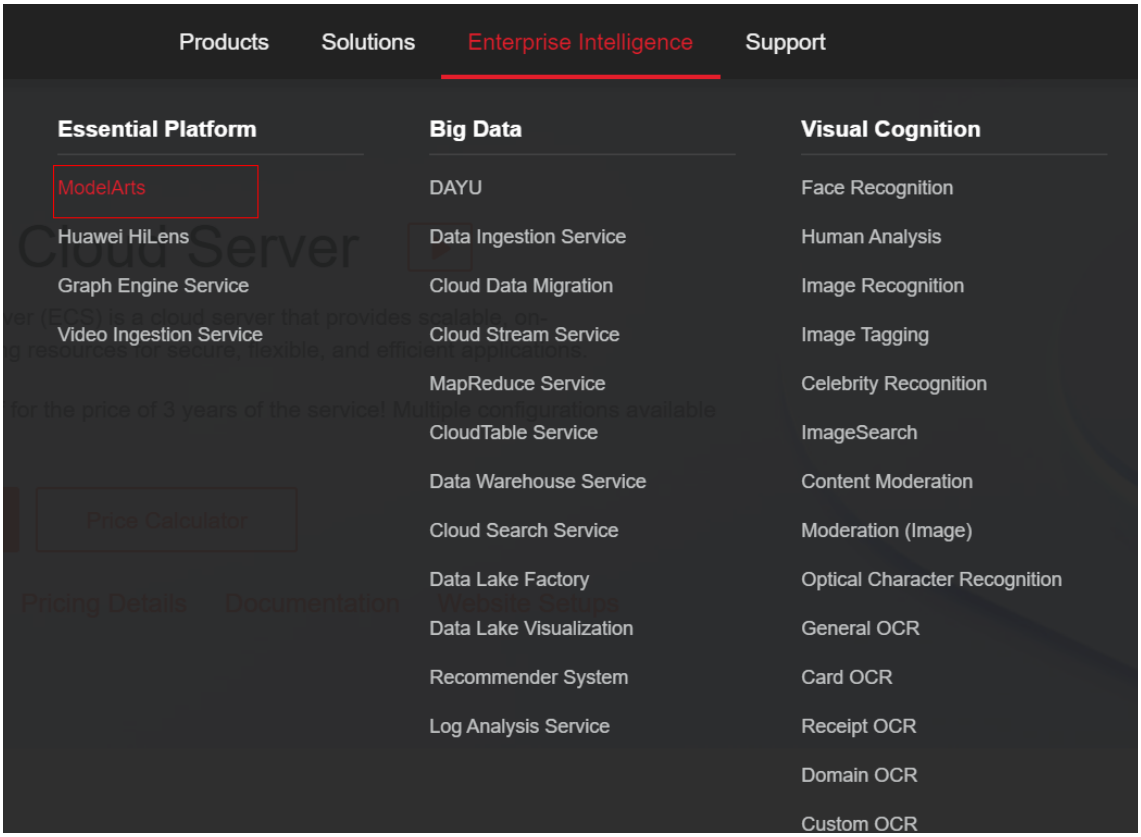


Figure 4-10

After accessing ModelArts, you can select multiple GPU-accelerated instances and various services provided by ModelArts, which will be demonstrated in Chapter 8.

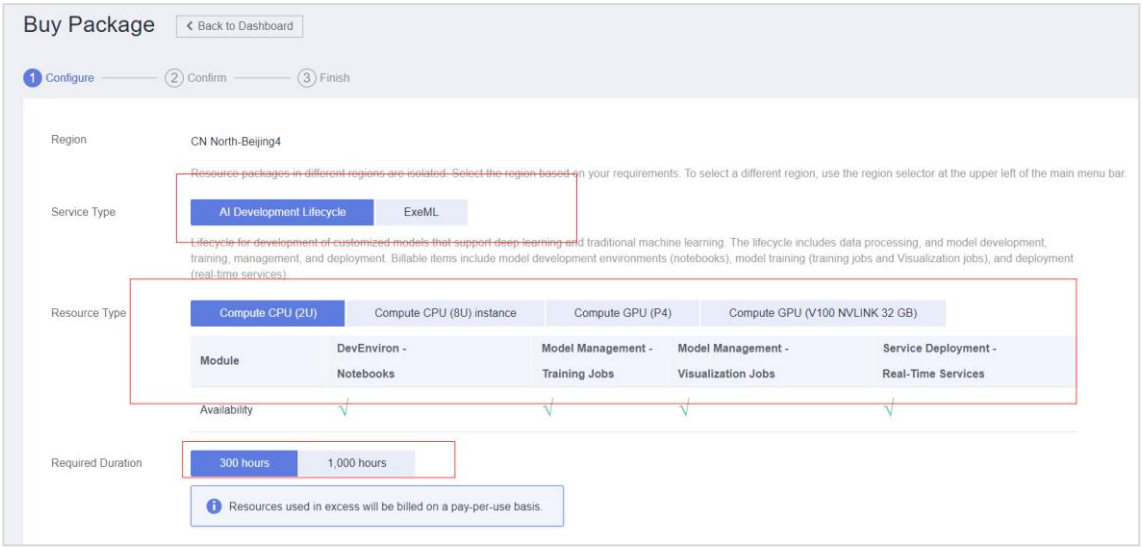


Figure 4-11

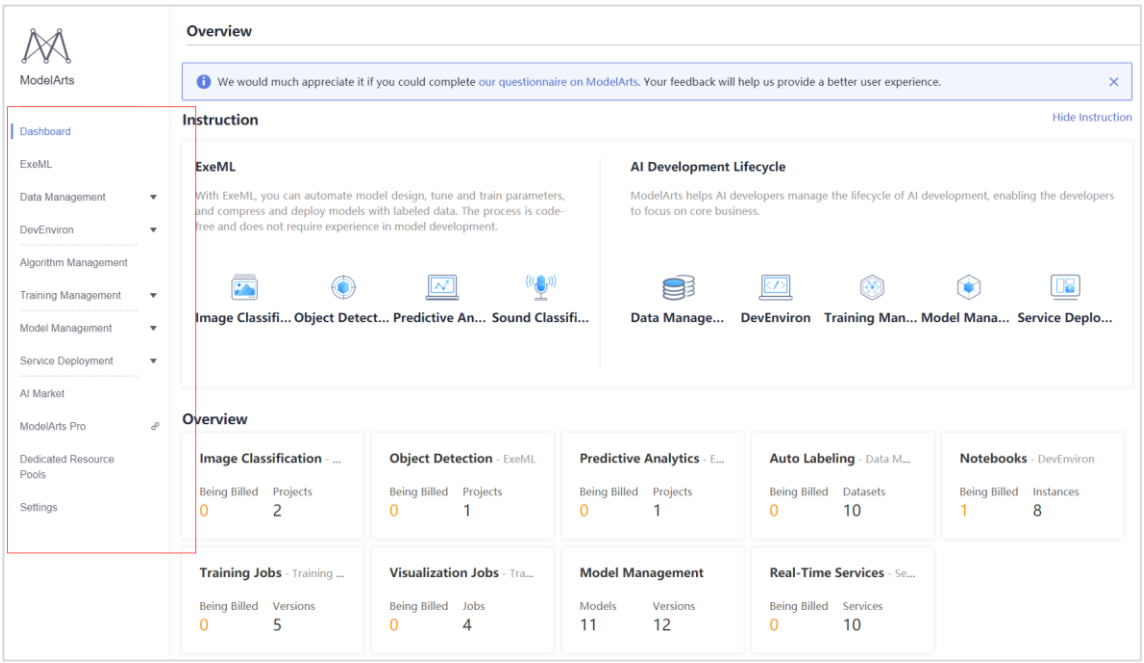


Figure 4-12

Huawei AI Certification Training

HCIA-AI

Machine Learning Experiment Guide

ISSUE:3.0

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

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

Notice

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

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

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

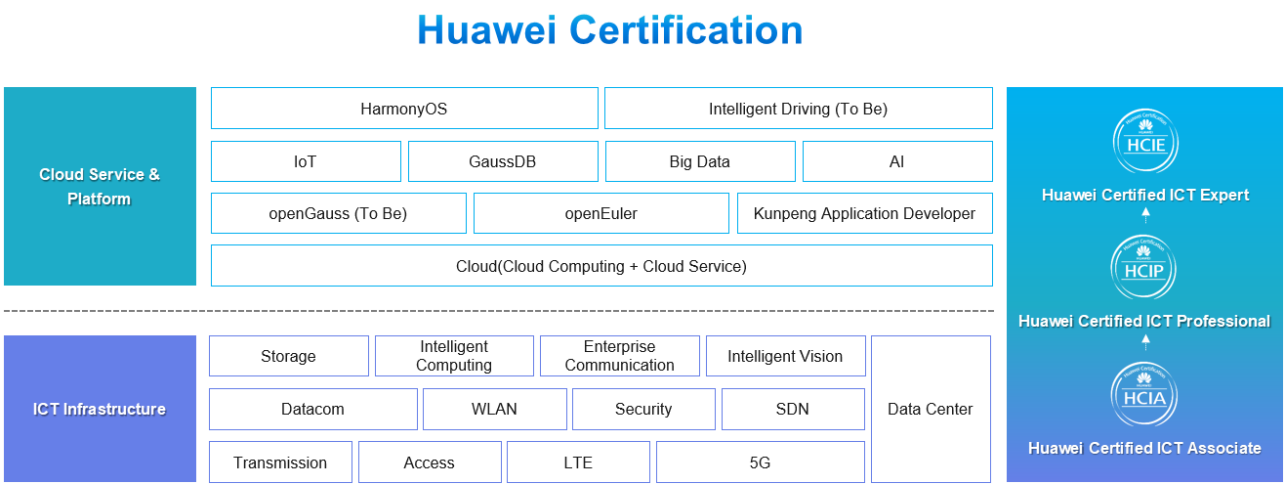
Huawei Certification is an integral part of the company's "Platform + Ecosystem" strategy, it supports the ICT infrastructure featuring "Cloud-Pipe-Device". It evolves to reflect the latest trends of ICT development. Huawei Certification consists of two categories: ICT Infrastructure, and Cloud Service & Platform.

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

With its leading talent development system and certification standards, Huawei is committed to developing ICT professionals in the digital era, building a healthy ICT talent ecosystem.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



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 machine learning programming.

Description

This guide contains one experiment, which is based on how to use sklearn-learn and python packages to predict house prices in Boston using different regression algorithms. It is hoped that trainees or readers can get started with machine learning and have the basic programming capability of machine learning building.

Background Knowledge Required

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

Experiment Environment Overview

Python Development Tool

This experiment environment is developed and compiled based on Python 3.6 and XGBoost will be used.

Contents

1 Detail of linear regression

1.1 Introduction

1.1.1 About This Experiment

This experiment mainly uses basic Python code and the simplest data to reproduce how a linear regression algorithm iterates and fits the existing data distribution step by step.

The experiment mainly used Numpy module and Matplotlib module.Numpy for calculation, Matplotlib for drawing.

1.1.2 Objectives

The main purpose of this experiment is as follows.

- Familiar with basic Python statements
- Master the implementation steps of linear regression

1.2 Experiment Code

1.2.1 Data preparation

10 data were randomly set, and the data were in a linear relationship.

The data is converted to array format so that it can be computed directly when multiplication and addition are used.

Code:

```
#Import the required modules, numpy for calculation, and Matplotlib for drawing
import numpy as np
import matplotlib.pyplot as plt
#This code is for jupyter Notebook only
%matplotlib inline

# define data, and change list to array
x = [3,21,22,34,54,34,55,67,89,99]
x = np.array(x)
y = [1,10,14,34,44,36,22,67,79,90]
y = np.array(y)

#Show the effect of a scatter plot
plt.scatter(x,y)
```

Output:

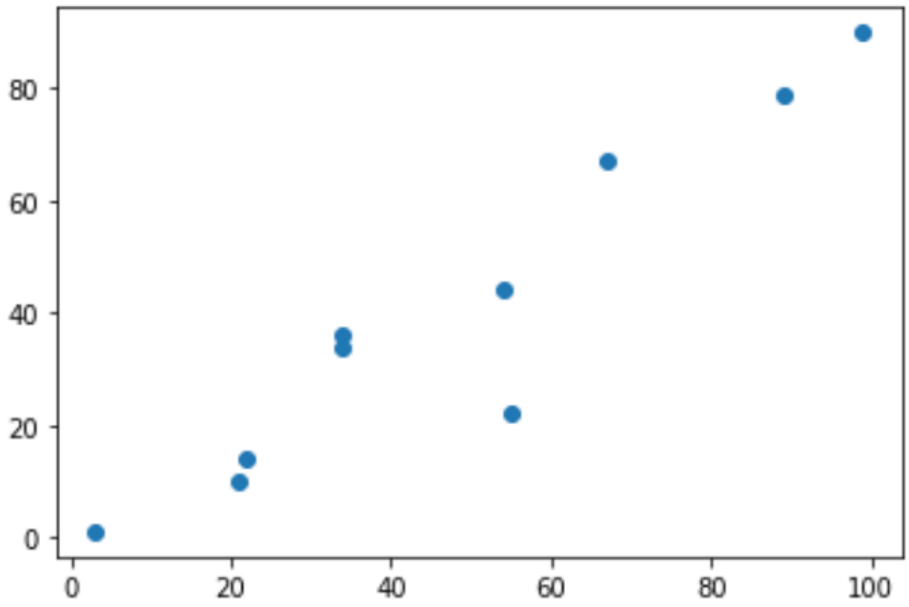


Figure 1-1 Scatter Plot

1.2.2 Define related functions

Model function: Defines a linear regression model wx+b.

Loss function: loss function of Mean square error.

Optimization function: gradient descent method to find partial derivatives of w and b.

Code:

```
#The basic linear regression model is wx+ b, and since this is a two-dimensional space, the model is ax+ b

def model(a, b, x):
    return a*x + b

#The most commonly used loss function of linear regression model is the loss function of mean variance difference
def loss_function(a, b, x, y):
    num = len(x)
    prediction=model(a,b,x)
    return (0.5/num) * (np.square(prediction-y)).sum()

#The optimization function mainly USES partial derivatives to update two parameters a and b
def optimize(a,b,x,y):
    num = len(x)
    prediction = model(a,b,x)
    #Update the values of A and B by finding the partial derivatives of the loss function on a and b
    da = (1.0/num) * ((prediction -y)*x).sum()
    db = (1.0/num) * ((prediction -y).sum())
```

```
    a = a - Lr*da
    b = b - Lr*db
    return a, b

#iterated function, return a and b
def iterate(a,b,x,y,times):
    for i in range(times):
        a,b = optimize(a,b,x,y)
    return a,b
```

1.2.3 Start the iteration

Step 1 Initialization and Iterative optimization model

Code:

```
#Initialize parameters and display
a = np.random.rand(1)
print(a)
b = np.random.rand(1)
print(b)
Lr = 1e-4

#For the first iteration, the parameter values, losses, and visualization after the iteration are displayed
a,b = iterate(a,b,x,y,1)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

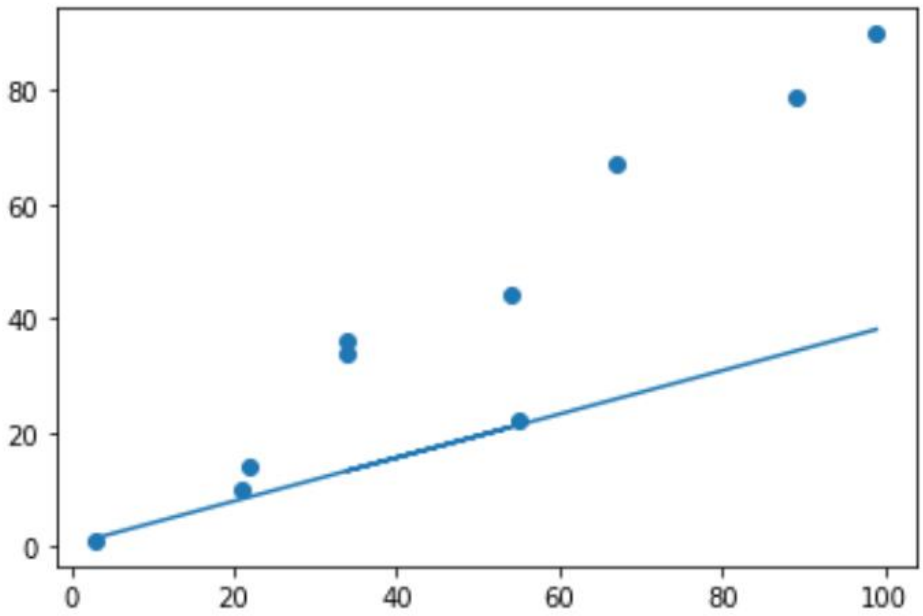


Figure 1-2 Iterate 1 time

Step 2 In the second iteration, the parameter values, loss values and visualization effects after the iteration are displayed

Code:

```
a,b = iterate(a,b,x,y,2)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

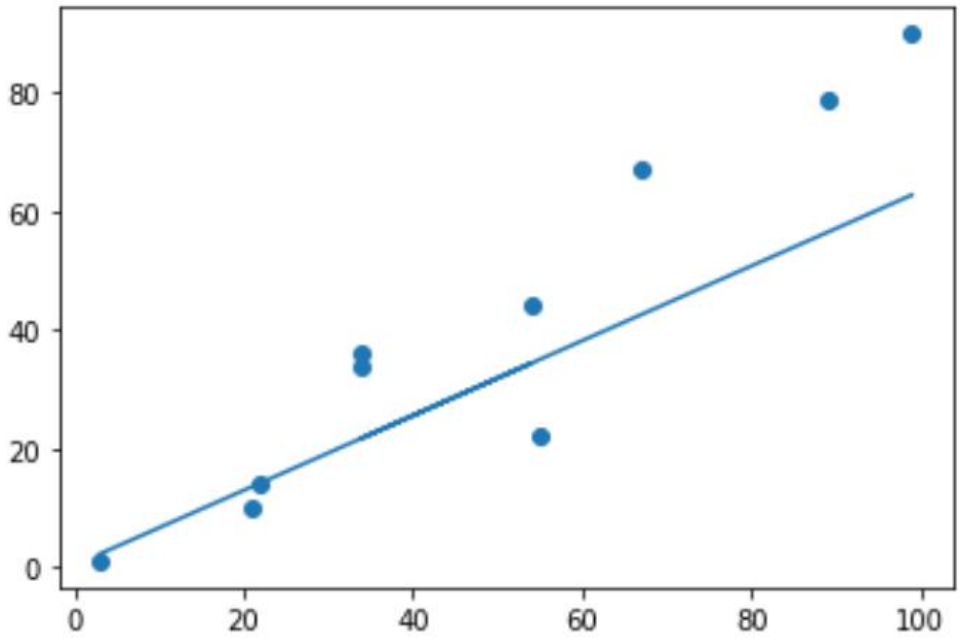


Figure 1-3 Iterate 2 times

Step 3 The third iteration shows the parameter values, loss values and visualization after iteration

Code:

```
a,b = iterate(a,b,x,y,3)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

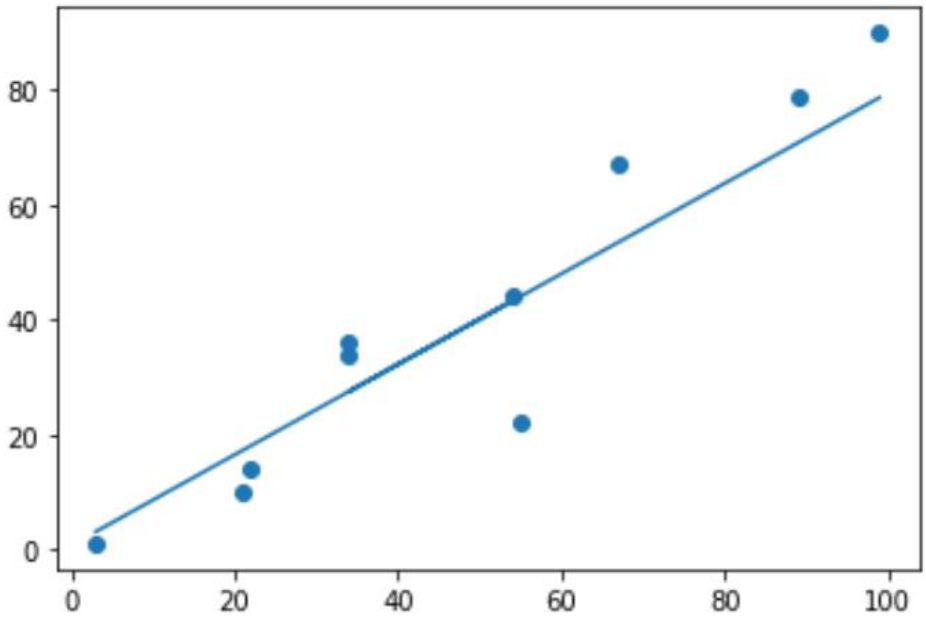


Figure 1-4 Iterate 3 times

Step 4 In the fourth iteration, parameter values, loss values and visualization effects are displayed

Code:

```
a,b = iterate(a,b,x,y,4)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

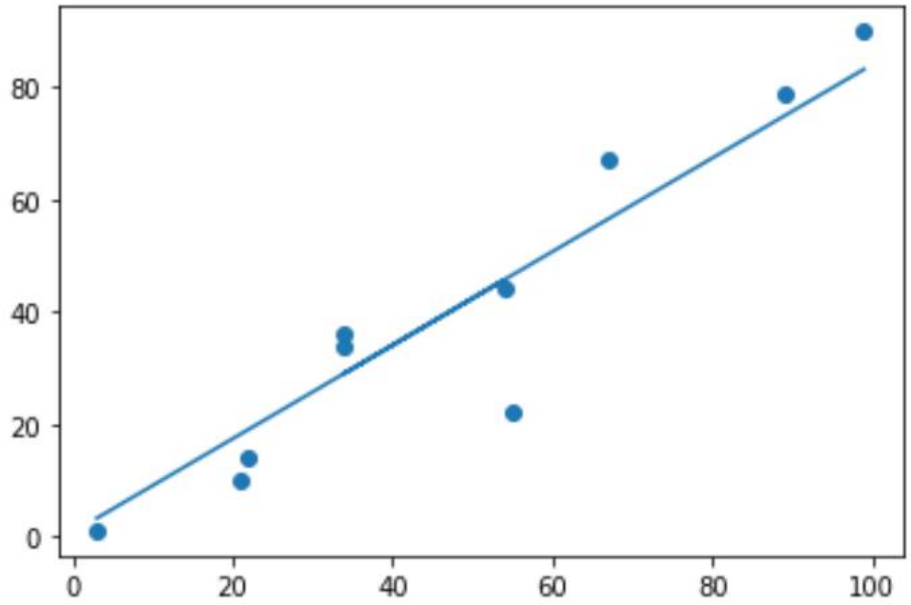


Figure 1-5 Iterate 4 times

Step 5 The fifth iteration shows the parameter value, loss value and visualization effect after iteration

Code:

```
a,b = iterate(a,b,x,y,5)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

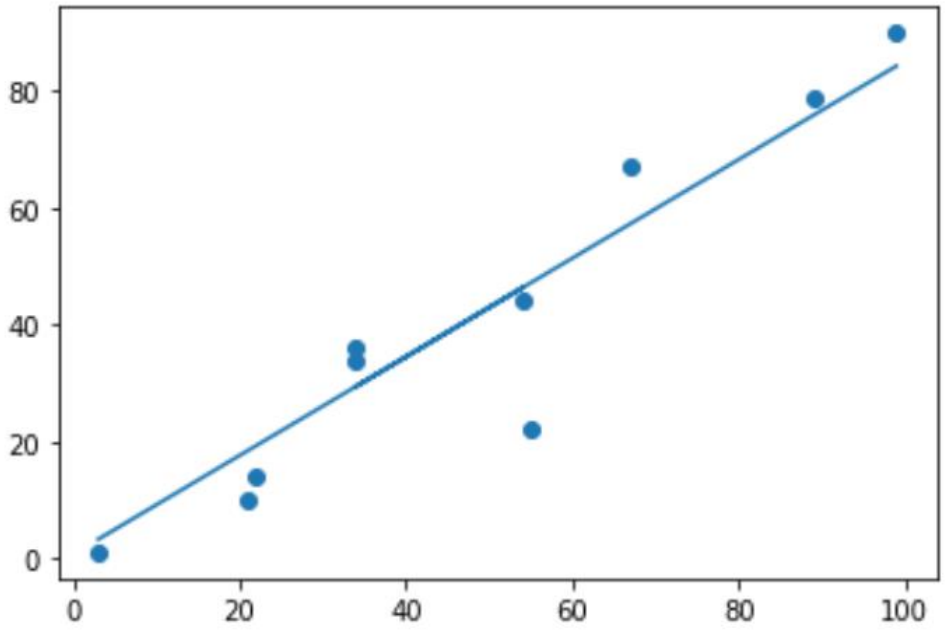


Figure 1-6 Iterate 5 times

Step 6 The 10000th iteration, showing the parameter values, losses and visualization after iteration

Code:

```
a,b = iterate(a,b,x,y,10000)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

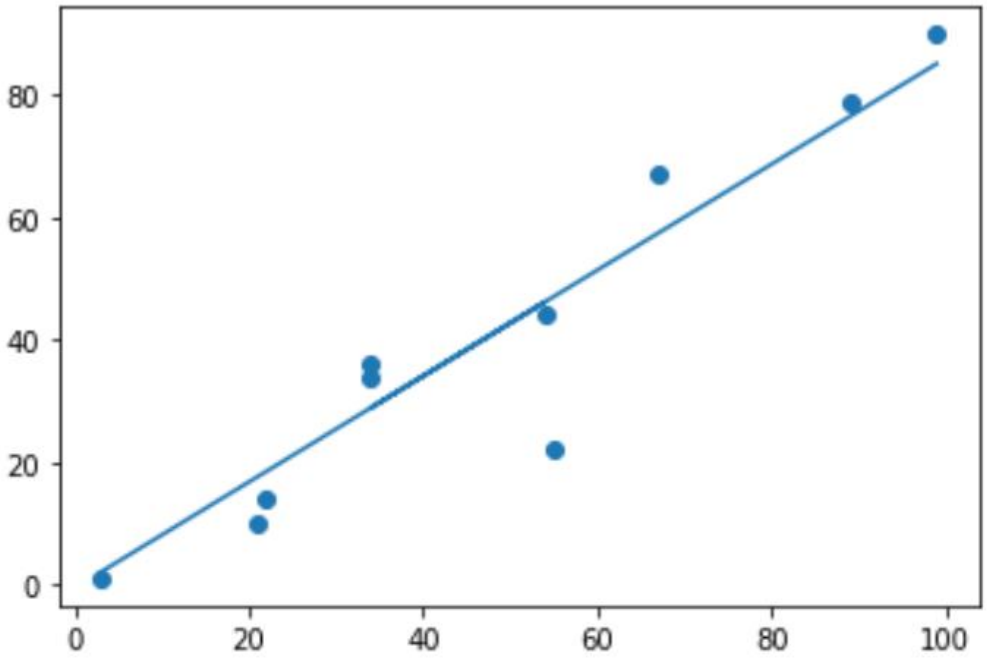


Figure 1-7 Iterate 10000 times

1.3 Thinking and practice

1.3.1 Question 1

Try to modify the original data yourself, Think about it: Does the loss value have to go to zero?

1.3.2 Question 2

Modify the values of Lr, Think: What is the role of the Lr parameter?

2 Decision tree details

2.1 Introduction

2.1.1 About This Experiment

This experiment focuses on the decision tree algorithm through the basic Python code.

It mainly uses Numpy module, Pandas module and Math module. We will implement the CART tree（Classification and Regressiontree models) in this experiment.

You have to download the dataset before this experiment through this link:

https://data-certification.obs.cn-east-2.myhuaweicloud.com/ENG/HCIA-AI/V3.0/ML-Dataset.rar

2.1.2 Objectives

The purpose of this experiment is as follows:

- Familiar with basic Python syntax
- Master the principle of Classification tree and implement with Python code
- Master the principle of Regression tree and implement with Python code

2.2 Experiment Code

2.2.1 Import the modules you need

Pandas is a tabular data processing module.

Math is mainly used for mathematical calculations.

Numpy is the basic computing module.

Code:

```
import pandas as pd
import math
import numpy as np
```

2.2.2 Superparameter definition section

Here you can choose to use Classification tree or Regression tree. Specifies the address of the dataset. Get feature name. Determine whether the algorithm matches the data set

Code:

```
algorithm = "Regression"    #    Algorithm: Classification, Regression
algorithm = "Classification"    #    Algorithm: Classification, Regression

# Dataset1: Text features and text labels
#df = pd.read_csv("D:/Code/Decision Treee/candidate/decision-trees-for-ml-master/decision-trees-for-ml-master/dataset/golf.txt")

# Dataset2: Mix features and Numeric labels, here you have to change the path to yours.
df = pd.read_csv("ML-Dataset/golf4.txt")

# This dictionary is used to store feature types of continuous numeric features and discrete literal features for subsequent judgment
dataset_features = dict()

num_of_columns = df.shape[1]-1
#The data type of each column of the data is saved for displaying the data name
for i in range(0, num_of_columns):
    #Gets the column name and holds the characteristics of a column of data by column
    column_name = df.columns[i]
    #Save the type of the data
    dataset_features[column_name] = df[column_name].dtypes
# The size of the indent when display
root = 1

# If the algorithm selects a regression tree but the label is not a continuous value, an error is reported
if algorithm == 'Regression':
    if df['Decision'].dtypes == 'object':
        raise ValueError('dataset wrong')
# If the tag value is continuous, the regression tree must be used
if df['Decision'].dtypes != 'object':
    algorithm = 'Regression'
    global_stdev = df['Decision'].std(ddof=0)
```

2.2.3 Define the functions required to complete the algorithm

Step 1 ProcessContinuousFeatures: Used to convert a continuous digital feature into a category feature.

Code:

```
# This function is used to handle numeric characteristics
def processContinuousFeatures(cdf, column_name, entropy):
    # Numerical features are arranged in order
    unique_values = sorted(cdf[column_name].unique())

    subset_ginis = [];
    subset_red_stdevs = []

    for i in range(0, len(unique_values) - 1):
        threshold = unique_values[i]
        # Find the segmentation result if the first number is used as the threshold
        subset1 = cdf[cdf[column_name] <= threshold]
```

```
            subset2 = cdf[cdf[column_name] > threshold]
            # Calculate the proportion occupied by dividing the two parts
            subset1_rows = subset1.shape[0];
            subset2_rows = subset2.shape[0]
            total_instances = cdf.shape[0]
            # In the text feature part, entropy is calculated by using the cycle,
            # and in the numeric part, entropy is calculated by using the two groups after segmentation,
            # and the degree of entropy reduction is obtained
            if algorithm == 'Classification':
                decision_for_subset1 = subset1['Decision'].value_counts().tolist()
                decision_for_subset2 = subset2['Decision'].value_counts().tolist()

                gini_subset1 = 1;
                gini_subset2 = 1

                for j in range(0, len(decision_for_subset1)):
                    gini_subset1 = gini_subset1 - math.pow((decision_for_subset1[j] / subset1_rows), 2)

                for j in range(0, len(decision_for_subset2)):
                    gini_subset2 = gini_subset2 - math.pow((decision_for_subset2[j] / subset2_rows), 2)

                gini = (subset1_rows / total_instances) * gini_subset1 + (subset2_rows / total_instances)
* gini_subset2

                subset_ginis.append(gini)

            # Take standard deviation as the judgment basis, calculate the decrease value of standard
deviation at this time
            elif algorithm == 'Regression':
                superset_stdev = cdf['Decision'].std(ddof=0)
                subset1_stdev = subset1['Decision'].std(ddof=0)
                subset2_stdev = subset2['Decision'].std(ddof=0)

                threshold_weighted_stdev = (subset1_rows / total_instances) * subset1_stdev + (
                        subset2_rows / total_instances) * subset2_stdev
                threshold_reducted_stdev = superset_stdev - threshold_weighted_stdev
                subset_red_stdevs.append(threshold_reducted_stdev)

        #Find the index of the split value
        if algorithm == "Classification":
            winner_one = subset_ginis.index(min(subset_ginis))
        elif algorithm == "Regression":
            winner_one = subset_red_stdevs.index(max(subset_red_stdevs))
        # Find the corresponding value according to the index
        winner_threshold = unique_values[winner_one]

        # Converts the original data column to an edited string column.
        # Characters smaller than the threshold are modified with the <= threshold value
        cdf[column_name] = np.where(cdf[column_name] <= winner_threshold, "<=" +
str(winner_threshold),">" + str(winner_threshold))

        return cdf
```

Step 2 CalculateEntropy: Used to calculate Gini or variances, they are the criteria for classifying.

Code:

```
# This function calculates the entropy of the column, and the input data must contain the Decision column
def calculateEntropy(df):
    # The regression tree entropy is 0
    if algorithm == 'Regression':
        return 0

    rows = df.shape[0]
    # Use Value_counts to get all values stored as dictionaries, keys: finds keys, and Tolist: change to lists.
    # This line of code finds the tag value.
    decisions = df['Decision'].value_counts().keys().tolist()

    entropy = 0
    # Here the loop traverses all the tags
    for i in range(0, len(decisions)):
        # Record the number of times the tag value appears
        num_of_decisions = df['Decision'].value_counts().tolist()[i]
        # probability of occurrence
        class_probability = num_of_decisions / rows
        # Calculate the entropy and sum it up
        entropy = entropy - class_probability * math.log(class_probability, 2)

    return entropy
```

Step 3 FindDecision: Find which feature in the current data to classify.

Code:

```
# The main purpose of this function is to traverse the entire column of the table,
# find which column is the best split column, and return the name of the column
def findDecision(ddf):
    # If it's a regression tree, then you take the standard deviation of the true value
    if algorithm == 'Regression':
        stdev = ddf['Decision'].std(ddof=0)
    # Get the entropy of the decision column
    entropy = calculateEntropy(ddf)

    columns = ddf.shape[1];
    rows = ddf.shape[0]
    # Used to store Gini and standard deviation values
    ginis = [];
    reducted_stdevs = []
    # Traverse all columns and calculate the relevant indexes of all columns according to algorithm selection
    for i in range(0, columns - 1):
        column_name = ddf.columns[i]
        column_type = ddf[column_name].dtypes

        # Determine if the column feature is a number, and if so, process the data using the following function,
        # which modifies the data to a string type category on return.
```

```
        # The idea is to directly use character characteristics, continuous digital characteristics into
discrete character characteristics
        if column_type != 'object':
            ddf = processContinuousFeatures(ddf, column_name, entropy)
        # The statistical data in this column can be obtained, and the continuous data can be
directly classified after processing,
        # and the categories are less than the threshold and greater than the threshold
        classes = ddf[column_name].value_counts()
        gini = 0;
        weighted_stdev = 0
        # Start the loop with the type of data in the column
        for j in range(0, len(classes)):
            current_class = classes.keys().tolist()[j]
            # The final classification result corresponding to the data is selected
            # by deleting the value of the df column equal to the current data
            subdataset = ddf[ddf[column_name] == current_class]

            subset_instances = subdataset.shape[0]
            # The entropy of information is calculated here
            if algorithm == 'Classification':   # GINI index
                decision_list = subdataset['Decision'].value_counts().tolist()

                subgini = 1

                for k in range(0, len(decision_list)):
                    subgini = subgini - math.pow((decision_list[k] / subset_instances), 2)

                gini = gini + (subset_instances / rows) * subgini
            # The regression tree is judged by the standard deviation,
            # and the standard deviation of the subclasses in this column is calculated here
            elif algorithm == 'Regression':
                subset_stdev = subdataset['Decision'].std(ddof=0)
                weighted_stdev = weighted_stdev + (subset_instances / rows) * subset_stdev

        # Used to store the final value of this column
        if algorithm == "Classification":
            ginis.append(gini)
        # Store the decrease in standard deviation for all columns
        elif algorithm == 'Regression':
            reducted_stdev = stdev - weighted_stdev
            reducted_stdevs.append(reducted_stdev)

    # Determine which column is the first branch
    # by selecting the index of the largest value from the list of evaluation indicators
    if algorithm == "Classification":
        winner_index = ginis.index(min(ginis))
    elif algorithm == "Regression":
        winner_index = reducted_stdevs.index(max(reducted_stdevs))
    winner_name = ddf.columns[winner_index]

    return winner_name
```

Step 4 FormatRule: Standardize the final output format.

Code:

```
# ROOT is a number used to generate ' 'to adjust the display format of the decision making process
def formatRule(root):
    resp = ''

    for i in range(0, root):
        resp = resp + '    '

    return resp
```

Step 5 BuildDecisionTree: Main function.

Code:

```
# With this function, you build the decision tree model,
# entering data in dataframe format, the root value, and the file address
# If the value in the column is literal, it branches directly by literal category
def buildDecisionTree(df, root):
    # Identify the different charForResp
    charForResp = '"'
    if algorithm == 'Regression':
        charForResp = ''

    tmp_root = root * 1

    df_copy = df.copy()
    # Output the winning column of the decision tree, enter a list,
    # and output the column name of the decision column in the list
    winner_name = findDecision(df)

    # Determines whether the winning column is a number or a character
    numericColumn = False
    if dataset_features[winner_name] != 'object':
        numericColumn = True

    # To ensure the integrity of the original data and prevent the data from changing,
    # mainly to ensure that the data of other columns besides the winning column does not change,
    # so as to continue the branch in the next step.
    columns = df.shape[1]
    for i in range(0, columns - 1):
        column_name = df.columns[i]
        if df[column_name].dtype != 'object' and column_name != winner_name:
            df[column_name] = df_copy[column_name]
    # Find the element in the branching column
    classes = df[winner_name].value_counts().keys().tolist()
    # Traversing all classes in the branch column has two functions:
    # 1. Display which class is currently traversed to; 2. Determine whether the current class is
already leaf node
    for i in range(0, len(classes)):
        # Find the Subdataset as in FindDecision, but discard this column of the current branch
        current_class = classes[i]
        subdataset = df[df[winner_name] == current_class]
        # At the same time, the data of the first branch column is discarded and the remaining data
is processed
        subdataset = subdataset.drop(columns=[winner_name])
```

```
        # Edit the display situation. If it is a numeric feature, the character conversion has been
completed when searching for branches.
        #If it is not a character feature, it is displayed with column names
        if numericColumn == True:
            compareTo = current_class    # current class might be <=x or >x in this case
        else:
            compareTo = " == '" + str(current_class) + "'"

        terminateBuilding = False

        # ----------------------------------------------

        # This determines whether it is already the last leaf node
        if len(subdataset['Decision'].value_counts().tolist()) == 1:
            final_decision = subdataset['Decision'].value_counts().keys().tolist()[
                0]    # all items are equal in this case
            terminateBuilding = True
        # At this time, only the Decision column is left, that is, all the segmentation features have
been used
        elif subdataset.shape[1] == 1:
            # get the most frequent one
            final_decision = subdataset['Decision'].value_counts().idxmax()
            terminateBuilding = True
        # The regression tree is judged as leaf node if the number of elements is less than 5
        #elif algorithm == 'Regression' and subdataset.shape[0] < 5:    # pruning condition
        # Another criterion is to take the standard deviation as the criterion and the sample mean in
the node as the value of the node
        elif algorithm == 'Regression' and subdataset['Decision'].std(ddof=0)/global_stdev < 0.4:
            # get average
            final_decision = subdataset['Decision'].mean()
            terminateBuilding = True
        # ----------------------------------------------
        # Here we begin to output the branching results of the decision tree.。

        print(formatRule(root), "if ", winner_name, compareTo, ":")

        # ----------------------------------------------
        # check decision is made
        if terminateBuilding == True:
            print(formatRule(root + 1), "return ", charForResp + str(final_decision) + charForResp)
        else:   # decision is not made, continue to create branch and leafs
            # The size of the indent at display represented by root
            root = root + 1
            # Call this function again for the next loop
            buildDecisionTree(subdataset, root)

        root = tmp_root * 1
```

2.2.4 Execute the code

Code:

```
# call the function
buildDecisionTree(df, root)
```

Output:

```
if Outlook == 'Sunny' :
    if Temp. <=83 :
        if Wind == 'Strong' :
            if Humidity <=95 :
                return 30
        if Wind == 'Weak' :
            return 36.5
    if Temp. >83 :
        return 25
if Outlook == 'Rain' :
    if Wind == 'Weak' :
        return 47.666666666666664
    if Wind == 'Strong' :
        return 26.5
if Outlook == 'Overcast' :
    return 46.25
```

Figure 2-1 Regression tree result

```
if Outlook == 'Sunny' :
    if Humidity == 'High' :
        return 'No'
    if Humidity == 'Normal' :
        return 'Yes'
if Outlook == 'Rain' :
    if Wind == 'Weak' :
        return 'Yes'
    if Wind == 'Strong' :
        return 'No'
if Outlook == 'Overcast' :
    return 'Yes'
```

Figure 2-2 CART tree result

Huawei AI Certification Training

HCIA-AI

Mainstream Development Framework

Lab Guide

Issue: 3.0

Huawei Technologies Co., Ltd.

Trademarks and Permissions

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

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

Notice

The purchased products, services, and features are stipulated by the commercial contract made between Huawei and the customer. All or partial products, services, and features described in this document may not be within the purchased scope or the usage scope. Unless otherwise agreed by 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 Certification System

Huawei Certification is an integral part of the company's "Platform + Ecosystem" strategy, it supports the ICT infrastructure featuring "Cloud-Pipe-Device". It evolves to reflect the latest trends of ICT development.

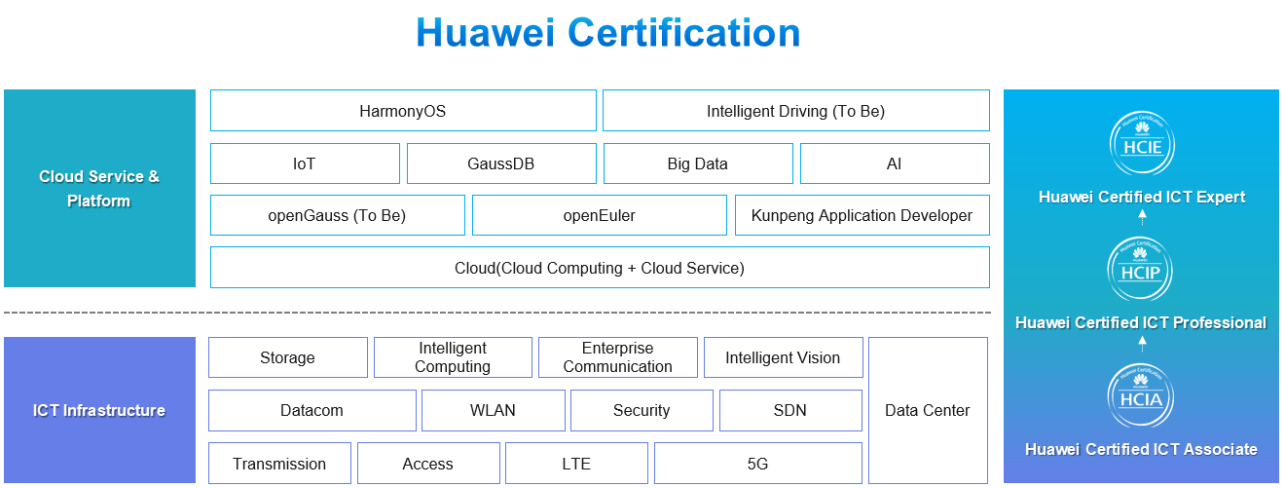
Huawei Certification consists of two categories: ICT Infrastructure, and Cloud Service & Platform. Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

With its leading talent development system and certification standards, Huawei is committed to developing ICT professionals in the digital era, building a healthy ICT talent ecosystem.

HCIA-AI V3.0 certification is intended for cultivating and conducting qualification of engineers who are capable of creatively designing and developing AI products and solutions using machine learning and deep learning algorithms.

HCIA-AI V3.0 certified engineers understand the development history of AI, Huawei Ascend AI system, and Huawei full-stack AI strategy in all scenarios, master traditional machine learning and deep learning algorithms, and are able to use the TensorFlow and MindSpore frameworks to build, train, and deploy neural networks. With this certificate, you are qualified for positions including sales, marketing, product manager, project management, and technical support in the AI field.

Huawei Certification Portfolio



About This Document

Introduction

This document is intended for trainees who are preparing for the HCIA-AI certification examination or readers who want to learn AI basics and TensorFlow programming basics.

Description

This lab guide includes the following three exercises:

- Exercise 1 mainly introduces the basic syntax of TensorFlow 2.
- Exercise 2 introduces common modules of TensorFlow 2, especially the Keras API.
- Exercise 3 is a handwritten font image recognition exercise. It uses basic code to help learners understand how to recognize handwritten fonts using TensorFlow 2.

Background Knowledge Required

This course is a basic course for Huawei certification. Before beginning this course, you should:

- Have basic Python knowledge
- Be familiar with basic concepts of TensorFlow
- Understand basic Python programming knowledge

Contents

1 TensorFlow 2 Basics

1.1 Introduction

1.1.1 About This Exercise

This exercise introduces tensor operations of TensorFlow 2, including tensor creation, slicing, indexing, tensor dimension modification, tensor arithmetic operations, and tensor sorting, to help you understand the basic syntax of TensorFlow 2.

1.1.2 Objectives

- Learn how to create tensors.
- Learn how to slice and index tensors.
- Master the syntax of tensor dimension changes.
- Master arithmetic operations of tensors.
- Know how to sort tensors.
- Understand eager execution and AutoGraph based on code.

1.2 Tasks

1.2.1 Introduction to Tensors

In TensorFlow, tensors are classified into constant and variable tensors.

- A defined constant tensor has an immutable value and dimension while a defined variable tensor has a variable value and an immutable dimension.
- In a neural network, a variable tensor is generally used as a matrix for storing weights and other information, and is a trainable data type. A constant tensor can be used as a variable for storing hyperparameters or other structural information.

1.2.1.1 Tensor Creation

1.2.1.1.1 Creating a Constant Tensor

Common methods for creating a constant tensor include:

- **tf.constant()**: creates a constant tensor.
- **tf.zeros(), tf.zeros_like(), tf.ones(),tf.ones_like()**: creates an all-zero or all-one constant tensor.
- **tf.fill()**: creates a tensor with a user-defined value.

- **tf.random**: creates a tensor with a known distribution.
- **tf.convert_to_tensor:** creates a list object by using NumPy and then converts it into a tensor.

Step 1 tf.constant()

tf.constant(value, dtype=None, shape=None, name='Const', verify_shape=False):

- **value**: value
- **dtype**: data type
- **shape**: tensor shape
- **name**: name for the constant tensor
- **verify_shape**: Boolean that enables verification of a shape of values. The default value is **False**. If **verify_shape** is set to **True**, the system checks whether the shape of **value** is consistent with **shape**. If they are inconsistent, an error is reported.

Code:

```
import tensorflow as tf
print(tf.__version__)
const_a = tf.constant([[1, 2, 3, 4]],shape=[2,2], dtype=tf.float32) # Create a 2x2 matrix with values 1, 2, 3, and 4.
const_a
```

Output:

```
2.0.0-beta1
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[1., 2.],
       [3., 4.]], dtype=float32)>
```

Code:

```
# View common attributes.
print("value of const_a: ", const_a.numpy())
print("data type of const_a: ", const_a.dtype)
print("shape of const_a: ", const_a.shape)
print("device that const_a will be generated: ", const_a.device)
```

Output:

```
The value of const_a is [[1. 2.]
 [3. 4.]]
The data type of const_a is <dtype: 'float32'>.
The shape of const_a is (2, 2).
const_a will be generated on /job:localhost/replica:0/task:0/device:CPU:0.
```

Step 2 tf.zeros(), tf.zeros_like(), tf.ones(),tf.ones_like()

The usage of **tf.ones()** and **tf.ones_like()** is similar to that of **tf.zeros()** and **tf.zeros_like()**. Therefore, the following describes how to use **tf.ones()** and **tf.ones_like()**.

Create a tensor with all elements set to zero.

tf.zeros(shape, dtype=tf.float32, name=None):

- **shape**: tensor shape
- **dtype**: type
- **name**: name for the operation

Code:

```
zeros_b = tf.zeros(shape=[2, 3], dtype=tf.int32) # Create a 2x3 matrix with all element values being 0.
```

Create a tensor with all elements set to zero based on the input tensor, with its shape being the same as that of the input tensor.
tf.zeros_like(input_tensor, dtype=None, name=None, optimize=True):

- **input_tensor**: tensor
- **dtype**: type
- **name**: name for the operation
- **optimize**: optimize or not

Code:

```
zeros_like_c = tf.zeros_like(const_a)
# View generated data.
zeros_like_c.numpy()
```

Output:

```
array([[0., 0.],
       [0., 0.]], dtype=float32)
```

Step 3 tf.fill()

Create a tensor and fill it with a specific value.
tf.fill(dims, value, name=None):

- **dims**: tensor shape, which is the same as the preceding shape
- **value**: tensor value
- **name**: name of the output

Code:

```
fill_d = tf.fill([3,3], 8) # 3x3 matrix with all element values being 8
# View data.
fill_d.numpy()
```

Output:

```
array([[8, 8, 8],
       [8, 8, 8],
       [8, 8, 8]], dtype=int32)
```

```
```

Step 4 tf.random

This module is used to generate a tensor with a specific distribution. The common methods in this module include **tf.random.uniform()**, **tf.random.normal()**, and **tf.random.shuffle()**. The following demonstrates how to use **tf.random.normal()**.

Create a tensor that conforms to the normal distribution.
tf.random.normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32,seed=None, name=None):

- **shape**: data shape
- **mean**: mean value of Gaussian distribution
- **stddev**: standard deviation of Gaussian distribution
- **dtype**: data type
- **seed**: random seed
- **name**: name for the operation

Code:

```
random_e = tf.random.normal([5,5],mean=0,stddev=1.0, seed = 1)
# View the created data.
random_e.numpy()
```

Output:

```
array([[-0.8521641 ,  2.0672443 , -0.94127315,  1.7840577 ,  2.9919195 ],
       [-0.8644102 ,  0.41812655, -0.85865736,  1.0617154 ,  1.0575105 ],
       [ 0.22457163, -0.02204755,  0.5084496 , -0.09113179, -1.3036906 ],
       [-1.1108295 , -0.24195422,  2.8516252 , -0.7503834 ,  0.1267275 ],
       [ 0.9460202 ,  0.12648873, -2.6540542 ,  0.0853276 ,  0.01731399]],
      dtype=float32)
```

Step 5 Create a list object by using NumPy and then convert it into a tensor by using **tf.convert_to_tensor**.

This method can convert the given value to a tensor. It converts Python objects of various types to Tensor objects.

tf.convert_to_tensor(value,dtype=None,dtype_hint=None,name=None):

- **value**: value to be converted
- **dtype**: tensor data type
- **dtype_hint**: optional element type for the returned tensor, used when **dtype** is **None**. In some cases, a caller may not have a dtype in mind when converting to a tensor, so dtype_hint can be used as a soft preference.

Code:

```
# Create a list.
list_f = [1,2,3,4,5,6]
```

```
# View the data type.
type(list_f)
```

Output:

```
list
```

Code:

```
tensor_f = tf.convert_to_tensor(list_f, dtype=tf.float32)
tensor_f
```

Output:

```
<tf.Tensor: shape=(6,), dtype=float32, numpy=array([1., 2., 3., 4., 5., 6.], dtype=float32)>
```

1.2.1.1.2 Creating a Variable Tensor

In TensorFlow, variables are created and tracked via the **tf.Variable** class. A **tf.Variable** represents a tensor whose value can be changed by running ops on it. Specific ops allow you to read and modify the values of this tensor.

Code:

```
# To create a variable, provide an initial value.
var_1 = tf.Variable(tf.ones([2,3]))
var_1
```

Output:

```
<tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
array([[1., 1., 1.],
       [1., 1., 1.]], dtype=float32)>
```

Code:

```
# Read the variable value.
Print("value of var_1: ",var_1.read_value())
# Assign a new value to the variable.
var_value_1=[[1,2,3],[4,5,6]]
var_1.assign(var_value_1)
Print("new value for var_1: ",var_1.read_value())
```

Output:

```
Value of var_1: tf.Tensor(
[[1. 1. 1.]
 [1. 1. 1.]], shape=(2, 3), dtype=float32)
New value for var_1: tf.Tensor(
[[1. 2. 3.]
 [4. 5. 6.]], shape=(2, 3), dtype=float32)
```

Code:

```
# Add a value to this variable.
var_1.assign_add(tf.ones([2,3]))
var_1
```

Output:

```
<tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
array([[2., 3., 4.],
       [5., 6., 7.]], dtype=float32)>
```

1.2.1.2 Tensor Slicing and Indexing

1.2.1.2.1 Slicing

Major slicing methods include:

- [start: end]: extracts a data slice from the start position to the end position of a tensor.
- [start :end :step] or [::step]: extracts a data slice at an interval of step from the start position to the end position of a tensor.
- [::-1]: slices data from the last element.
- '...': indicates a data slice of any length.

Code:

```
# Create a 4-dimensional tensor. The tensor contains four images. The size of each image is 100 x
100 x 3.
tensor_h = tf.random.normal([4,100,100,3])
tensor_h
```

Output:

```
<tf.Tensor: shape=(4, 100, 100, 3), dtype=float32, numpy=
array([[[[ 1.68444023e-01, -7.46562362e-01, -4.34964240e-01],
         [-4.69263226e-01,  6.26460612e-01,  1.21065331e+00],
         [ 7.21675277e-01,  4.61057723e-01, -9.20868576e-01],
         ...,
```

Code:

```
# Extract the first image.
tensor_h[0,:,:,:]
```

Output:

```
<tf.Tensor: shape=(100, 100, 3), dtype=float32, numpy=
array([[[ 1.68444023e-01, -7.46562362e-01, -4.34964240e-01],
        [-4.69263226e-01,  6.26460612e-01,  1.21065331e+00],
        [ 7.21675277e-01,  4.61057723e-01, -9.20868576e-01],
        ...,
```

Code:

```
# Extract one slice every two images.
tensor_h[::2,...]
```

Output:

```
<tf.Tensor: shape=(2, 100, 100, 3), dtype=float32, numpy=
array([[[[ 1.68444023e-01, -7.46562362e-01, -4.34964240e-01],
         [-4.69263226e-01,  6.26460612e-01,  1.21065331e+00],
         [ 7.21675277e-01,  4.61057723e-01, -9.20868576e-01],
         ...,
```

Code:

```
# Slice data from the last element.
tensor_h[::-1]
```

Output:

```
<tf.Tensor: shape=(4, 100, 100, 3), dtype=float32, numpy=
array([[[[-1.70684665e-01,  1.52386248e+00, -1.91677585e-01],
         [-1.78917408e+00, -7.48436213e-01,  6.10363662e-01],
         [ 7.64770031e-01,  6.06725179e-02,  1.32704067e+00],
         ...,
```

1.2.1.2.2 Indexing

The basic format of an index is a[d1][d2][d3].

Code:

```
# Obtain the pixel in the [20,40] position in the second channel of the first image.
tensor_h[0][19][39][1]
```

Output:

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.38231283>
```

If the indexes to be extracted are nonconsecutive, **tf.gather** and **tf.gather_nd** are commonly used for data extraction in TensorFlow.

To extract data from a particular dimension:

tf.gather(params, indices,axis=None):

- **params**: input tensor
- **indices**: index of the data to be extracted
- **axis**: dimension of the data to be extracted

Code:

```
# Extract the first, second, and fourth images from tensor_h ([4,100,100,3]).
indices = [0,1,3]
```

```
tf.gather(tensor_h,axis=0,indices=indices)
```

Output:

```
<tf.Tensor: shape=(3, 100, 100, 3), dtype=float32, numpy=
array([[[[ 1.68444023e-01, -7.46562362e-01, -4.34964240e-01],
         [-4.69263226e-01,  6.26460612e-01,  1.21065331e+00],
         [ 7.21675277e-01,  4.61057723e-01, -9.20868576e-01],
         ...,
```

tf.gather_nd allows data extraction from multiple dimensions:

tf.gather_nd(params,indices):

- **params**: input tensor
- **indices**: index of the data to be extracted. Generally, this is a multidimensional list.

Code:

```
# Extract the pixel in [1,1] in the first dimension of the first image and pixel in [2,2] in the first dimension of the second image in tensot_h ([4,100,100,3]).
indices = [[0,1,1,0],[1,2,2,0]]
tf.gather_nd(tensor_h,indices=indices)
```

Output:

```
<tf.Tensor: shape=(2,), dtype=float32, numpy=array([0.5705869, 0.9735735], dtype=float32)>
```

1.2.1.3 Tensor Dimension Modification

1.2.1.3.1 Dimension Display

Code:

```
const_d_1 = tf.constant([[1, 2, 3, 4]],shape=[2,2], dtype=tf.float32)
# Three common methods for displaying a dimension:
print(const_d_1.shape)
print(const_d_1.get_shape())
print(tf.shape(const_d_1))# The output is a tensor. The value of the tensor indicates the size of the tensor dimension to be displayed.
```

Output:

```
(2, 2)
(2, 2)
tf.Tensor([2 2], shape=(2,), dtype=int32)
```

As described above, **.shape** and **.get_shape()** return TensorShape objects, while **tf.shape(x)** returns Tensor objects.

1.2.1.3.2 Dimension Reshaping

tf.reshape(tensor,shape,name=None):

- **tensor**: input tensor

- **shape**: shape of the reshaped tensor

Code:

```
reshape_1 = tf.constant([[1,2,3],[4,5,6]])
print(reshape_1)
tf.reshape(reshape_1, (3,2))
```

Output:

```
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[1, 2],
       [3, 4],
       [5, 6]], dtype=int32)>
```

1.2.1.3.3 Dimension Expansion

tf.expand_dims(input,axis,name=None):

- **input**: input tensor
- **axis**: adds a dimension after the axis dimension. Given an input of D dimensions, **axis** must be in range [-(D+1), D] (inclusive). A negative value indicates the reverse order.

Code:

```
# Generate a 100 x 100 x 3 tensor to represent a 100 x 100 three-channel color image.
expand_sample_1 = tf.random.normal([100,100,3], seed=1)
print("original data size: ",expand_sample_1.shape)
Print("add a dimension (axis=0) before the first dimension: ",tf.expand_dims(expand_sample_1, axis=0).shape)
Print("add a dimension (axis=1) before the second dimension: ",tf.expand_dims(expand_sample_1, axis=1).shape)
Print("add a dimension (axis=-1) after the last dimension: ",tf.expand_dims(expand_sample_1, axis=-1).shape)
```

Output:

```
Original data size: (100, 100, 3)
Add a dimension (axis=0) before the first dimension: (1, 100, 100, 3)
Add a dimension (axis=1) before the second dimension: (100, 1, 100, 3)
Add a dimension (axis=-1) after the last dimension: (100, 100, 3, 1)
```

1.2.1.3.4 Dimension Squeezing

tf.squeeze(input,axis=None,name=None):

This method is used to remove dimensions of size 1 from the shape of a tensor.

- **input**: input tensor
- **axis**: If you don not want to remove all size 1 dimensions, remove specific size 1 dimensions by specifying **axis**.

Code:

```
# Generate a 100 x 100 x 3 tensor.
orig_sample_1 = tf.random.normal([1,100,100,3])
```

```
print("original data size: ",orig_sample_1.shape)
squeezed_sample_1 = tf.squeeze(orig_sample_1)
print("squeezed data size: ",squeezed_sample_1.shape)

# The dimension of 'squeeze_sample_2' is [1, 2, 1, 3, 1, 1].
squeeze_sample_2 = tf.random.normal([1, 2, 1, 3, 1, 1])
t_1 = tf.squeeze(squeeze_sample_2) # Dimensions of size 1 are removed.
print('t_1.shape:', t_1.shape)
# Remove a specific dimension:
# 't' is a tensor of shape [1, 2, 1, 3, 1, 1]
t_1_new = tf.squeeze(squeeze_sample_2, [2, 4])
print('t_1_new.shape:', t_1_new.shape)
```

```
Output:
Original data size: (1, 100, 100, 3)
Squeezed data size: (100, 100, 3)
t_1.shape: (2, 3)
t_1_new.shape: (1, 2, 3, 1)
```

1.2.1.3.5 Transpose

tf.transpose(a,perm=None,conjugate=False,name='transpose'):

- **a**: input tensor
- **perm**: permutation of the dimensions of **a**, generally used to transpose high-dimensional arrays
- **conjugate**: conjugate transpose
- **name**: name for the operation

Code:

```
# Low-dimensional transposition is simple. Input the tensor to be transposed by calling tf.transpose.
trans_sample_1 = tf.constant([1,2,3,4,5,6],shape=[2,3])
print("original data size: ",trans_sample_1.shape)
transposed_sample_1 = tf.transpose(trans_sample_1)
print("transposed data size: ",transposed_sample_1.shape)
```

Output:

```
Original data size: (2, 3)
Transposed data size: (3, 2)
```

Code:

```
The perm parameter is required for transposing high-dimensional data. perm indicates the permutation of the dimensions of the input tensor.
For a three-dimensional tensor, its original dimension permutation is [0, 1, 2] (perm), indicating the length, width, and height of the high-dimensional data, respectively.
By changing the value sequence in perm, you can transpose the corresponding dimension of the data.
# Generate a 4 x 100 x 200 x 3 tensor to represent four 100 x 200 three-channel color images.
```

```
trans_sample_2 = tf.random.normal([4,100,200,3])
print("original data size: ",trans_sample_2.shape)
# Exchange the length and width of the four images. The value range of perm is changed from [0,1,2,3] to [0,2,1,3].
transposed_sample_2 = tf.transpose(trans_sample_2,[0,2,1,3])
print("transposed data size: ",transposed_sample_2.shape)
```

Output:

```
Original data size: (4, 100, 200, 3)
Transposed data size: (4, 200, 100, 3)
```

1.2.1.3.6 Broadcast (broadcast_to)

broadcast_to is used to broadcast data from a low dimension to a high dimension.

tf.broadcast_to(input,shape,name=None):

- **input**: input tensor
- **shape**: size of the output tensor

Code:

```
broadcast_sample_1 = tf.constant([1,2,3,4,5,6])
print("original data: ",broadcast_sample_1.numpy())
broadcasted_sample_1 = tf.broadcast_to(broadcast_sample_1,shape=[4,6])
print("broadcast data: ",broadcasted_sample_1.numpy())
```

Output:

```
Original data: [1 2 3 4 5 6]
Broadcast data: [[1 2 3 4 5 6]
 [1 2 3 4 5 6]
 [1 2 3 4 5 6]
 [1 2 3 4 5 6]]
```

Code:

```
# During the operation, if two arrays have different shapes, TensorFlow automatically triggers the broadcast mechanism as NumPy does.
a = tf.constant([[ 0, 0, 0],
                 [10,10,10],
                 [20,20,20],
                 [30,30,30]])
b = tf.constant([1,2,3])
print(a + b)
```

```
Output:
tf.Tensor(
[[ 1  2  3]
 [11 12 13]
 [21 22 23]
 [31 32 33]], shape=(4, 3), dtype=int32)
```

1.2.1.4 Arithmetic Operations on Tensors

1.2.1.4.1 Arithmetic Operators

Arithmetic operations include addition (**tf.add**), subtraction **(tf.subtract**), multiplication (**tf.multiply**), division (**tf.divide**), logarithm (**tf.math.log**), and powers (**tf.pow**). The following is an example of addition.

Code:

```
a = tf.constant([[3, 5], [4, 8]])
b = tf.constant([[1, 6], [2, 9]])
print(tf.add(a, b))
```

Output:

```
tf.Tensor(
[[ 4 11]
 [ 6 17]], shape=(2, 2), dtype=int32)
```

1.2.1.4.2 Matrix Multiplication

Matrix multiplication is implemented by calling **tf.matmul**.

Code:

```
tf.matmul(a,b)
```

Output:

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[13, 63],
       [20, 96]], dtype=int32)>
```

1.2.1.4.3 Tensor Statistics Collection

Methods for collecting tensor statistics include:

- **tf.reduce_min/max/mean()**: calculates the minimum, maximum, and mean values.
- **tf.argmax()/tf.argmin()**: calculates the positions of the maximum and minimum values.
- **tf.equal()**: checks whether two tensors are equal by element.
- **tf.unique()**: removes duplicate elements from a tensor.
- **tf.nn.in_top_k(prediction, target, K)**: calculates whether the predicted value is equal to the actual value and returns a tensor of the Boolean type.

The following demonstrates how to use **tf.argmax()**.

Return the subscript of the maximum value.

tf.argmax(input,axis):

- **input**: input tensor
- **axis**: The maximum value is output based on the axis dimension.

Code:

```
argmax_sample_1 = tf.constant([[1,3,2],[2,5,8],[7,5,9]])
print("input tensor: ",argmax_sample_1.numpy())
max_sample_1 = tf.argmax(argmax_sample_1, axis=0)
max_sample_2 = tf.argmax(argmax_sample_1, axis=1)
print("locate the maximum value by column: ",max_sample_1.numpy())
print("locate the maximum value by row: ",max_sample_2.numpy())
```

Output:

```
Input tensor: [1 3 2]
 [2 5 8]
 [7 5 9]]
Locate the maximum value by column: [2 1 2].
Locate the maximum value by row: [1 2 2].
```

1.2.1.5 Dimension-based Arithmetic Operations

In TensorFlow, operations such as **tf.reduce_*** reduce tensor dimensions. These operations can be performed on the dimension elements of a tensor, for example, calculating the mean value by row and calculating a product of all elements in the tensor.

Common operations include **tf.reduce_sum** (addition), **tf.reduce_prod** (multiplication), **tf.reduce_min** (minimum), **tf.reduce_max** (maximum), **tf.reduce_mean** (mean), **tf.reduce_all** (logical AND), **tf.reduce_any** (logical OR), and **tf.reduce_logsumexp** (**log(sum(exp))**).

The methods of using these operations are similar. The following uses the **tf.reduce_sum** operation as an example.

Compute the sum of elements across dimensions of a tensor.

tf.reduce_sum(input_tensor, axis=None, keepdims=False,name=None):

- **input_tensor**: tensor to reduce
- **axis**: axis to be calculated. If this parameter is not specified, the mean value of all elements is calculated.
- **keepdims**: whether to reduce the dimension. If this parameter is set to **True**, the output result retains the shape of the input tensor. If this parameter is set to **False**, the dimension of the output result is reduced.
- **name**: name for the operation

Code:

```
reduce_sample_1 = tf.constant([1,2,3,4,5,6],shape=[2,3])
print("original data",reduce_sample_1.numpy())
print("compute the sum of all elements in a tensor (axis=None):
",tf.reduce_sum(reduce_sample_1,axis=None).numpy())
print("compute the sum of each column by column (axis=0):
",tf.reduce_sum(reduce_sample_1,axis=0).numpy())
print("compute the sum of each column by row (axis=1):
",tf.reduce_sum(reduce_sample_1,axis=1).numpy())
```

Output:

```
Original data [1 2 3]
 [4 5 6]]
Compute the sum of all elements in the tensor (axis=None): 21
Compute the sum of each column (axis=0): [5 7 9]
Compute the sum of each column (axis=1): [6 15]
```

1.2.1.6 Tensor Concatenation and Splitting

1.2.1.6.1 Tensor Concatenation

In TensorFlow, tensor concatenation operations include:

- **tf.contact()**: concatenates tensors along one dimension. Other dimensions remain unchanged.
- **tf.stack()**: stacks the tensor list of rank R into a tensor of rank (R+1). Dimensions are changed after stacking.

tf.concat(values, axis, name='concat'):

- **values**: input tensor
- **axis**: dimension along which to concatenate
- **name**: name for the operation

Code:

```
concat_sample_1 = tf.random.normal([4,100,100,3])
concat_sample_2 = tf.random.normal([40,100,100,3])
Print("original data size: ",concat_sample_1.shape,concat_sample_2.shape)
concated_sample_1 = tf.concat([concat_sample_1,concat_sample_2],axis=0)
print("concatenated data size: ",concated_sample_1.shape)
```

Output:

```
Original data size: (4, 100, 100, 3) (40, 100, 100, 3)
Concatenated data size: (44, 100, 100, 3)
```

A dimension is added to an original matrix in the same way. **axis** determines the position where the dimension is added.

tf.stack(values, axis=0, name='stack'):

- **values**: a list of tensor objects with the same shape and type
- **axis**: axis to stack along
- **name**: name for the operation

Code:

```
stack_sample_1 = tf.random.normal([100,100,3])
stack_sample_2 = tf.random.normal([100,100,3])
Print("original data size: ",stack_sample_1.shape, stack_sample_2.shape)
# Dimension addition after concatenating. If axis is set to 0, a dimension is added before the first dimension.
stacked_sample_1 = tf.stack([stack_sample_1, stack_sample_2],axis=0)
print("concatenated data size: ",stacked_sample_1.shape)
```

Output:

```
Original data size: (100, 100, 3) (100, 100, 3)
Concatenated data size: (2, 100, 100, 3)
```

1.2.1.6.2 Tensor Splitting

In TensorFlow, tensor splitting operations include:

- **tf.unstack()**: unpacks tensors along the specific dimension.
- **tf.split()**: splits a tensor into a list of sub tensors based on specific dimensions.

Compared with **tf.unstack()**, **tf.split()** is more flexible.

tf.unstack(value,num=None,axis=0,name='unstack'):

- **value**: input tensor
- **num**: outputs a list containing **num** elements. **num** must be equal to the number of elements in the specified dimension. Generally, this parameter is ignored.
- **axis**: axis to unstack along
- **name**: name for the operation

Code:

```
# Unpack data along the first dimension and output the unpacked data in a list.
tf.unstack(stacked_sample_1,axis=0)
```

Output:

```
[<tf.Tensor: shape=(100, 100, 3), dtype=float32, numpy=
 array([[[ 0.0665694 ,   0.7110351 ,   1.907618   ],
         [ 0.84416866,   1.5470593 , -0.5084871 ],
         [-1.9480026 , -0.9899087 , -0.09975405],
         ...,
```

tf.split(value, num_or_size_splits, axis=0):

- **value**: input tensor
- **num_or_size_splits**: number of splits
- **axis**: dimension along which to split

tf.split() can be split in either of the following ways:

1. If **num_or_size_splits** is an integer, the tensor is evenly split into several small tensors along the **axis=D** dimension.
2. If **num_or_size_splits** is a vector, the tensor is split into several smaller tensors based on the element values of the vector along the **axis=D** dimension.

Code:

```
import numpy as np
split_sample_1 = tf.random.normal([10,100,100,3])
print("original data size: ",split_sample_1.shape)
splited_sample_1 = tf.split(split_sample_1, num_or_size_splits=5,axis=0)
print("If m_or_size_splits is 5, the size of the split data is: ",np.shape(splited_sample_1))
```

```
splited_sample_2 = tf.split(split_sample_1, num_or_size_splits=[3,5,2],axis=0)
print("If num_or_size_splits is [3,5,2], the sizes of the split data are:",
      np.shape(splited_sample_2[0]),
      np.shape(splited_sample_2[1]),
      np.shape(splited_sample_2[2]))
```

Output:

```
Original data size: (10, 100, 100, 3)
If m_or_size_splits is 5, the size of the split data is (5, 2, 100, 100, 3).
If num_or_size_splits is [3,5,2], the sizes of the split data are (3, 100, 100, 3) (5, 100, 100, 3) (2, 100, 100, 3).
```

1.2.1.7 Tensor Sorting

In TensorFlow, tensor sorting operations include:

- **tf.sort()**: sorts tensors in ascending or descending order and returns the sorted tensors.
- **tf.argsort()**: sorts tensors in ascending or descending order and returns the indices.
- **tf.nn.top_k()**: returns the k largest values.

tf.sort/argsort(input, direction, axis):

- **input**: input tensor
- **direction**: direction in which to sort the values. The value can be **DESCENDING** or **ASCENDING**. The default value is **ASCENDING**.
- **axis**: axis along which to sort The default value is **-1**, which sorts the last axis.

Code:

```
sort_sample_1 = tf.random.shuffle(tf.range(10))
print("input tensor: ",sort_sample_1.numpy())
sorted_sample_1 = tf.sort(sort_sample_1, direction="ASCENDING")
print("tensor sorted in ascending order: ",sorted_sample_1.numpy())
sorted_sample_2 = tf.argsort(sort_sample_1,direction="ASCENDING")
print("index of elements in ascending order: ",sorted_sample_2.numpy())
```

Output:

```
Input tensor: [1 8 7 9 6 5 4 2 3 0]
Tensor sorted in ascending order: [0 1 2 3 4 5 6 7 8 9]
Index of elements in ascending order: [9 0 7 8 6 5 4 2 1 3]
```

tf.nn.top_k(input,K,sorted=TRUE):

- **input**: input tensor
- **K**: k largest values to be output and their indices
- **sorted**: **sorted=TRUE** indicates in ascending order. **sorted=FALSE** indicates in descending order.

Two tensors are returned:

- **values**: k largest values in each row

- **indices**: indices of **values** within the last dimension of **input**

Code:

```
values, index = tf.nn.top_k(sort_sample_1,5)
print("input tensor: ",sort_sample_1.numpy())
print("k largest values in ascending order: ", values.numpy())
print("indices of the k largest values in ascending order: ", index.numpy())
```

Output:

```
Input tensor: [1 8 7 9 6 5 4 2 3 0]
The k largest values in ascending order: [9 8 7 6 5]
Indices of the k largest values in ascending order: [3 1 2 4 5]
```

1.2.2 Eager Execution Mode of TensorFlow 2

Eager execution mode:

The eager execution mode of TensorFlow is a type of imperative programming, which is the same as the native Python. When you perform a particular operation, the system immediately returns a result.

Graph mode:

TensorFlow 1 adopts the graph mode to first build a computational graph, enable a session, and then feed actual data to obtain a result.

In eager execution mode, code debugging is easier, but the code execution efficiency is lower.

The following implements simple multiplication by using TensorFlow to compare the differences between the eager execution mode and the graph mode.

Code:

```
x = tf.ones((2, 2), dtype=tf.dtypes.float32)
y = tf.constant([[1, 2],
                 [3, 4]], dtype=tf.dtypes.float32)
z = tf.matmul(x, y)
print(z)
```

Output:

```
tf.Tensor(
[[4. 6.]
 [4. 6.]], shape=(2, 2), dtype=float32)
```

Code:

```
# Use the syntax of TensorFlow 1.x in TensorFlow 2.x. You can install the v1 compatibility package in TensorFlow 2 to inherit the TensorFlow 1.x code and disable the eager execution mode.
import tensorflow.compat.v1 as tf
tf.disable_eager_execution()
# Create a graph and define it as a computational graph.
a = tf.ones((2, 2), dtype=tf.dtypes.float32)
```

```
b = tf.constant([[1, 2],
                 [3, 4]], dtype=tf.dtypes.float32)
c = tf.matmul(a, b)
# Start a session and perform the multiplication operation to obtain data.
with tf.Session() as sess:
    print(sess.run(c))
```

Output:

```
[[4. 6.]
 [4. 6.]]
```

Restart the kernel to restore TensorFlow to version 2 and enable the eager execution mode. Another advantage of the eager execution mode lies in availability of native Python functions, such as the following condition statement:

Code:

```
import tensorflow as tf
thre_1 = tf.random.uniform([], 0, 1)
x = tf.reshape(tf.range(0, 4), [2, 2])
print(thre_1)
if thre_1.numpy() > 0.5:
    y = tf.matmul(x, x)
else:
    y = tf.add(x, x)
```

Output:

```
tf.Tensor(0.11304152, shape=(), dtype=float32)
```

With the eager execution mode, this dynamic control flow can generate a NumPy value extractable by a tensor, without using operators such as **tf.cond** and **tf.while** provided in the graph mode.

1.2.3 AutoGraph of TensorFlow 2

When used to comment out a function, the **tf.function** decorator can be called like any other function. **tf.function** will be compiled into a graph, so that it can run more efficiently on a GPU or TPU. In this case, the function becomes an operation in TensorFlow. The function can be directly called to output a return value. However, the function is executed in graph mode and the intermediate variable values cannot be directly viewed.

Code:

```
@tf.function
def simple_nn_layer(w,x,b):
    print(b)
    return tf.nn.relu(tf.matmul(w, x)+b)


w = tf.random.uniform((3, 3))
```

```
x = tf.random.uniform((3, 3))
b = tf.constant(0.5, dtype='float32')

simple_nn_layer(w,x,b)
```

Output:

```
Tensor("b:0", shape=(), dtype=float32)
<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
array([[1.4121541 , 1.1626956 , 1.2527422 ],
       [1.2903953 , 1.0956903 , 1.1309073 ],
       [1.1039395 , 0.92851776, 1.0752096 ]], dtype=float32)>
```

According to the output result, the value of **b** in the function cannot be directly viewed, but the return value can be viewed using **.numpy()**.

The following compares the performance of the graph and eager execution modes by performing the same operation (LSTM computation of one layer).

Code:

```
# Use the timeit module to measure the execution time of a small code segment.
import timeit
# Create a convolutional layer.
CNN_cell = tf.keras.layers.Conv2D(filters=100,kernel_size=2,strides=(1,1))

# Use @tf.function to convert the operation into a graph.
@tf.function
def CNN_fn(image):
    return CNN_cell(image)

image = tf.zeros([100, 200, 200, 3])

# Compare the execution time of the two modes.
CNN_cell(image)
CNN_fn(image)
# Call timeit.timeit to measure the time required for executing the code 10 times.
print("time required for performing the computation of one convolutional neural network (CNN)
layer in eager execution mode:", timeit.timeit(lambda: CNN_cell(image), number=10))
print("time required for performing the computation of one CNN layer in graph mode:",
timeit.timeit(lambda: CNN_fn(image), number=10))
```

Output:

```
Time required for performing the computation of one CNN layer in eager execution mode:
18.26327505100926
Time required for performing the computation of one CNN layer in graph mode: 6.740775318001397
```

The comparison shows that the code execution efficiency in graph mode is much higher. Therefore, the **@tf.function** can be used to improve the code execution efficiency.

2 Common Modules of TensorFlow 2

2.1 Introduction

This section describes the common modules of TensorFlow 2, including:

- **tf.data**: implements operations on datasets.

  These operations include reading datasets from the memory, reading CSV files, reading TFRecord files, and augmenting data.
- **tf.image**: implements image processing.

  These operations include image luminance transformation, saturation transformation, image size transformation, image rotation, and edge detection.
- **tf.gfile**: implements operations on files.

  These operations include reading, writing, and renaming files, and operating folders.
- **tf.keras**: a high-level API used to build and train deep learning models.
- **tf.distributions** and other modules

This section focuses on the **tf.keras** module to lay a foundation for deep learning modeling.

2.2 Objectives

Upon completion of this exercise, you will be able to master the common deep learning modeling interfaces of **tf.keras**.

2.3 Tasks

2.3.1 Model Building

2.3.1.1 Stacking a Model (tf.keras.Sequential)

The most common way to build a model is to stack layers by using **tf.keras.Sequential**.

Code:

```
import tensorflow as tf
print(tf.__version__)
print(tf.keras.__version__)
import tensorflow.keras.layers as layers
model = tf.keras.Sequential()
```

```
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Output:

```
2.0.0-beta1
2.2.4-tf
```

2.3.1.2 Building a Functional Model

Functional models are mainly built by using **tf.keras.Input** and **tf.keras.Model**, which are more complex than **tf.keras.Sequential** but have a good effect. Variables can be input at the same time or in different phases, and data can be output in different phases. Functional models are preferred if more than one model output is needed.

Stacked model (.Sequential) vs. functional model (.Model):
The **tf.keras.Sequential** model is a simple stack of layers that cannot represent arbitrary models. You can use the Keras functional API to build complex model topologies, for example:

- Multi-input models
- Multi-output models
- Models with shared layers
- Models with non-sequential data flows (for example, residual connections)

Code:

```
# Use the output of the current layer as the input of the next layer.
x = tf.keras.Input(shape=(32,))
h1 = layers.Dense(32, activation='relu')(x)
h2 = layers.Dense(32, activation='relu')(h1)
y = layers.Dense(10, activation='softmax')(h2)
model_sample_2 = tf.keras.models.Model(x, y)

# Print model information.
model_sample_2.summary()
```

Output:

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32)]              0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_4 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_5 (Dense)              (None, 10)                330
=================================================================
Total params: 2,442
```

```
Trainable params: 2,442
Non-trainable params: 0
_________________________________________________________________
```

2.3.1.3 Building a Network Layer (tf.keras.layers)

The **tf.keras.layers** module is used to configure neural network layers. Common classes include:

- **tf.keras.layers.Dense**: builds a fully connected layer.
- **tf.keras.layers.Conv2D**: builds a two-dimensional convolutional layer.
- **tf.keras.layers.MaxPooling2D/AveragePooling2D**: builds a maximum/average pooling layer.
- **tf.keras.layers.RNN**: Builds a recurrent neural network layer.
- **tf.keras.layers.LSTM/tf.keras.layers.LSTMCell**: builds an LSTM network layer/LSTM unit.
- **tf.keras.layers.GRU/tf.keras.layers.GRUCell**: builds a GRU unit/GRU network layer.
- **tf.keras.layers.Embedding**: converts a positive integer (subscript) into a vector of a fixed size, for example, converts [[4],[20]] into [[0.25,0.1],[0.6,-0.2]]. The embedding layer can be used only as the first model layer.
- **tf.keras.layers.Dropout**: builds the dropout layer.

The following describes **tf.keras.layers.Dense**, **tf.keras.layers.Conv2D**, **tf.keras.layers.MaxPooling2D/AveragePooling2D**, and **tf.keras.layers.LSTM/tf.keras.layers.LSTMCell**.

The main network configuration parameters in **tf.keras.layers** include:

- **activation**: sets the activation function of a layer. By default, the system does not use any activation functions.
- **kernel_initializer** and **bias_initializer**: initialization schemes that create a layer's weights (kernel and bias). This defaults to the Glorot uniform initializer.
- **kernel_regularizer** and **bias_regularizer**: regularization schemes that apply to a layer's weights (kernel and bias), for example, L1 or L2 regularization. By default, the system does not use regularization functions.

2.3.1.3.1 tf.keras.layers.Dense

Main configuration parameters in **tf.keras.layers.Dense** include:

- **units**: number of neurons
- **activation**: activation function
- **use_bias**: whether to use the bias terms. Bias terms are used by default.
- **kernel_initializer**: initialization scheme that creates a layer's weight (kernel)
- **bias_initializer**: initialization scheme that creates a layer's weight (bias)
- **kernel_regularizer**: regularization scheme that applies a layer's weight (kernel)
- **bias_regularizer**: regularization scheme that applies a layer's weight (bias)
- **activity_regularizer**: regular item applied to the output, a Regularizer object
- **kernel_constraint**: constraint applied to a weight

- **bias_constraint**: constraint applied to a weight

Code:

```
# Create a fully connected layer that contains 32 neurons and set the activation function to sigmoid.
# The activation parameter can be set to a function name string, for example, sigmoid, or a function object, for example, tf.sigmoid.
layers.Dense(32, activation='sigmoid')
layers.Dense(32, activation=tf.sigmoid)

# Set kernel_initializer.
layers.Dense(32, kernel_initializer=tf.keras.initializers.he_normal)
# Set kernel_regularizer to the L2 regularization.
layers.Dense(32, kernel_regularizer=tf.keras.regularizers.l2(0.01))
```

Output:

```
<TensorFlow.python.keras.layers.core.Dense at 0x130c519e8>
```

2.3.1.3.2 tf.keras.layers.Conv2D

Main configuration parameters in **tf.keras.layers.Conv2D** include:

- **filters**: number of convolution kernels (output dimensions)
- **kernel_size**: width and length of a convolution kernel
- **strides**: convolution stride
- **padding**: zero padding policy

  When **padding** is set to **valid**, only valid convolution is performed, that is, boundary data is not processed. When **padding** is set to **same**, the convolution result at the boundary is reserved, and consequently, the output shape is usually the same as the input shape.
- **activation**: activation function
- **data_format**: data format. The value can be **channels_first** or **channels_last**. For example, for a 128 x 128 RGB image, data is organized as (3,128,128) if the value is **channels_first**, and as (128,128,3) if the value is **channels_last**. The default value of this parameter is the value defined in **~/.keras/keras.json**. If the value has never been set, the default value is **channels_last**.
- Other parameters include **use_bias**, **kernel_initializer**, **bias_initializer**, **kernel_regularizer**, **bias_regularizer**, **activity_regularizer**, **kernel_constraints**, and **bias_constraints**.

Code:

```
layers.Conv2D(64,[1,1],2,padding='same',activation="relu")
```

Output:

```
<TensorFlow.python.keras.layers.convolutional.Conv2D at 0x106c510f0>
```

2.3.1.3.3 tf.keras.layers.MaxPooling2D/AveragePooling2D

Main configuration parameters in **tf.keras.layers.MaxPooling2D/AveragePooling2D** include:

- **pool_size**: size of the pooled kernel. For example, if the matrix (2, 2) is used, the image becomes half of the original length in both dimensions. If this parameter is set to an integer, the integer is the value of all dimensions.
- **strides**: stride value.
- Other parameters include **padding** and **data_format**.

Code:

```
layers.MaxPooling2D(pool_size=(2,2),strides=(2,1))
```

Output:

```
<TensorFlow.python.keras.layers.pooling.MaxPooling2D at 0x132ce1f98>
```

2.3.1.3.4 tf.keras.layers.LSTM/tf.keras.layers.LSTMCell

Main configuration parameters in **tf.keras.layers.LSTM/tf.keras.layers.LSTMCell** include:

- **units**: output dimension
- **input_shape** (**timestep**, **input_dim**): **timestep** can be set to **None**, and **input_dim** indicates the input data dimensions.
- **activation**: activation function
- **recurrent_activation**: activation function to use for the recurrent step
- **return_sequences**: If the value is **True**, all sequences are returned. If the value is **False**, the output in the last cell of the output sequence is returned.
- **return_state**: Boolean value, indicating whether to return the last state in addition to the output.
- **dropout**: float between 0 and 1, fraction of the neurons to drop for the linear transformation of the inputs
- **recurrent_dropout**: float between 0 and 1, fraction of the neurons to drop for the linear transformation of the recurrent state

Code:

```
import numpy as np
inputs = tf.keras.Input(shape=(3, 1))
lstm = layers.LSTM(1, return_sequences=True)(inputs)
model_lstm_1 = tf.keras.models.Model(inputs=inputs, outputs=lstm)

inputs = tf.keras.Input(shape=(3, 1))
lstm = layers.LSTM(1, return_sequences=False)(inputs)
model_lstm_2 = tf.keras.models.Model(inputs=inputs, outputs=lstm)


# Sequences t1, t2, and t3
data = [[[0.1],
   [0.2],
```

```
   [0.3]]]
print(data)
print("output when return_sequences is set to True",model_lstm_1.predict(data))
print("output when return_sequences is set to False",model_lstm_2.predict(data))
```

Output:

```
[[[0.1], [0.2], [0.3]]]
Output when return_sequences is set to True: [[[-0.0106758 ]
   [-0.02711176]
   [-0.04583194]]]
Output when return_sequences is set to False: [[0.05914127]]
```

LSTMcell is the implementation unit of the LSTM layer.

- LSTM is an LSTM network layer.
- LSTMCell is a single-step computing unit, that is, an LSTM unit.

```
#LSTM
tf.keras.layers.LSTM(16, return_sequences=True)

#LSTMCell
x = tf.keras.Input((None, 3))
y = layers.RNN(layers.LSTMCell(16))(x)
model_lstm_3= tf.keras.Model(x, y)
```

2.3.2 Training and Evaluation

2.3.2.1 Model Compilation

After a model has been built, call **compile** to configure its learning process:

- compile(optimizer='rmsprop', loss=None, metrics=None, loss_weights=None):
- **optimizer**: optimizer
- **loss**: loss function, cross entropy for binary tasks and MSE for regression tasks
- **metrics**: model evaluation criteria during training and testing. For example, **metrics** can be set to **['accuracy']**. To specify multiple evaluation criteria, transfer a dictionary. For example, set **metrics** to **{'output_a':'accuracy'}**.
- **loss_weights**: If the model has multiple task outputs, you need to specify a weight for each output when optimizing the global loss.

Code:

```
model = tf.keras.Sequential()
model.add(layers.Dense(10, activation='softmax'))
# Determine the optimizer, loss function, and model evaluation method (metrics).
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss=tf.keras.losses.categorical_crossentropy,
              metrics=[tf.keras.metrics.categorical_accuracy])
```

2.3.2.2 Model Training

fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0, validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None, validation_steps=None):

- **x**: input training data
- **y**: target (labeled) data
- **batch_size**: number of samples for each gradient update. The default value **32**.
- **epochs**: number of iteration rounds of the training model
- **verbose**: log display mode. The value can be **0**, **1**, or **2**.

  0 = no display

  1 = progress bar

  2 = one line for each round
- **callbacks**: callback function used during training
- **validation_split**: ratio of the validation dataset to the training data
- **validation_data**: validation dataset. This parameter overwrites **validation_split**.
- **shuffle**: whether to shuffle data before each round of iteration. This parameter is invalid when **steps_per_epoch** is not **None**.
- **initial_epoch**: epoch at which to start training (useful for resuming a previous training weight)
- **steps_per_epoch**: set to the dataset size or **batch_size**
- **validation_steps**: Total number of steps (batches of samples) to be validated before stopping. This parameter is valid only when **steps_per_epoch** is specified.

Code:

```
import numpy as np

train_x = np.random.random((1000, 36))
train_y = np.random.random((1000, 10))

val_x = np.random.random((200, 36))
val_y = np.random.random((200, 10))

model.fit(train_x, train_y, epochs=10, batch_size=100,
          validation_data=(val_x, val_y))
```

Output:

```
Train on 1000 samples, validate on 200 samples
Epoch 1/10
1000/1000 [==============================] - 0s 488us/sample - loss: 12.6024 -
categorical_accuracy: 0.0960 - val_loss: 12.5787 - val_categorical_accuracy: 0.0850
Epoch 2/10
1000/1000 [==============================] - 0s 23us/sample - loss: 12.6007 -
categorical_accuracy: 0.0960 - val_loss: 12.5776 - val_categorical_accuracy: 0.0850
Epoch 3/10
```

```
1000/1000 [==============================] - 0s 31us/sample - loss: 12.6002 -
categorical_accuracy: 0.0960 - val_loss: 12.5771 - val_categorical_accuracy: 0.0850
...
Epoch 10/10
1000/1000 [==============================] - 0s 24us/sample - loss: 12.5972 -
categorical_accuracy: 0.0960 - val_loss: 12.5738 - val_categorical_accuracy: 0.0850

<TensorFlow.python.keras.callbacks.History at 0x130ab5518>
```

For large datasets, you can use **tf.data** to build training input pipelines.

Code:

```
dataset = tf.data.Dataset.from_tensor_slices((train_x, train_y))
dataset = dataset.batch(32)
dataset = dataset.repeat()
val_dataset = tf.data.Dataset.from_tensor_slices((val_x, val_y))
val_dataset = val_dataset.batch(32)
val_dataset = val_dataset.repeat()

model.fit(dataset, epochs=10, steps_per_epoch=30,
          validation_data=val_dataset, validation_steps=3)
```

Output:

```
Train for 30 steps, validate for 3 steps
Epoch 1/10
30/30 [==============================] - 0s 15ms/step - loss: 12.6243 - categorical_accuracy:
0.0948 - val_loss: 12.3128 - val_categorical_accuracy: 0.0833
...
30/30 [==============================] - 0s 2ms/step - loss: 12.5797 - categorical_accuracy:
0.0951 - val_loss: 12.3067 - val_categorical_accuracy: 0.0833
<TensorFlow.python.keras.callbacks.History at 0x132ab48d0>
```

2.3.2.3 Callback Functions

A callback function is an object passed to the model to customize and extend the model's behavior during training. You can customize callback functions or use embedded functions in **tf.keras.callbacks**. Common embedded callback functions include:
tf.keras.callbacks.ModelCheckpoint: periodically saves models.
tf.keras.callbacks.LearningRateScheduler: dynamically changes the learning rate.
tf.keras.callbacks.EarlyStopping: stops the training in advance.
tf.keras.callbacks.TensorBoard: uses the TensorBoard.

Code:

```
import os
# Set hyperparameters.
Epochs = 10
logdir=os.path.join("logs")
if not os.path.exists(logdir):
    os.mkdir(logdir)
# Define a function for dynamically setting the learning rate.
def lr_Scheduler(epoch):
```

```
    if epoch > 0.9 * Epochs:
        lr = 0.0001
    elif epoch > 0.5 * Epochs:
        lr = 0.001
    elif epoch > 0.25 * Epochs:
        lr = 0.01
    else:
        lr = 0.1

    print(lr)
    return lr


callbacks = [
  # Early stopping:
    tf.keras.callbacks.EarlyStopping(
        # Metric for determining whether the model performance has no further improvement
        monitor='val_loss',
        # Threshold for determining whether the model performance has no further improvement
        min_delta=1e-2,
        # Number of epochs in which the model performance has no further improvement
        patience=2),

    # Periodically save models.
     tf.keras.callbacks.ModelCheckpoint(
        # Model path
        filepath='testmodel_{epoch}.h5',
        # Determine whether to save the optimal model.
        save_best_only=True,
        monitor='val_loss'),

    # Dynamically change the learning rate.
    tf.keras.callbacks.LearningRateScheduler(lr_Scheduler),

    # Use the TensorBoard.
    tf.keras.callbacks.TensorBoard(log_dir=logdir)
]
model.fit(train_x, train_y, batch_size=16, epochs=Epochs,callbacks=callbacks, validation_data=(val_x,
val_y))
```

Output:

```
Train on 1000 samples, validate on 200 samples
0
0.1
Epoch 1/10
1000/1000 [==============================] - 0s 155us/sample - loss: 12.7907 -
categorical_accuracy: 0.0920 - val_loss: 12.7285 - val_categorical_accuracy: 0.0750
1
0.1
Epoch 2/10
1000/1000 [==============================] - 0s 145us/sample - loss: 12.6756 -
categorical_accuracy: 0.0940 - val_loss: 12.8673 - val_categorical_accuracy: 0.0950
...
0.001
```

```
Epoch 10/10
1000/1000 [==============================] - 0s 134us/sample - loss: 12.3627 -
categorical_accuracy: 0.1020 - val_loss: 12.3451 - val_categorical_accuracy: 0.0900

<TensorFlow.python.keras.callbacks.History at 0x133d35438>
```

2.3.2.4 Evaluation and Prediction

Evaluation and prediction functions: **tf.keras.Model.evaluate** and **tf.keras.Model.predict**.

Code:

```
# Model evaluation
test_x = np.random.random((1000, 36))
test_y = np.random.random((1000, 10))
model.evaluate(test_x, test_y, batch_size=32)
```

Output:

```
1000/1000 [==============================] - 0s 45us/sample - loss: 12.2881 -
categorical_accuracy: 0.0770
[12.288104843139648, 0.077]
```

Code:

```
# Model prediction
pre_x = np.random.random((10, 36))
result = model.predict(test_x,)
print(result)
```

Output:

```
[[0.04431767 0.24562006 0.05260926 ... 0.1016549   0.13826898 0.15511878]
 [0.06296062 0.12550288 0.07593573 ... 0.06219672 0.21190381 0.12361749]
 [0.07203944 0.19570401 0.11178136 ... 0.05625525 0.20609994 0.13041474]
 ...
 [0.09224506 0.09908539 0.13944311 ... 0.08630784 0.15009451 0.17172746]
 [0.08499582 0.17338121 0.0804626  ... 0.04409525 0.27150458 0.07133815]
 [0.05191234 0.11740112 0.08346355 ... 0.0842929   0.20141983 0.19982798]]
```

2.3.3 Model Saving and Restoration

2.3.3.1 Saving and Restoring an Entire Model

Code:

```
import numpy as np
# Save models.
logdir='./model'
if not os.path.exists(logdir):
    os.mkdir(logdir)

model.save(logdir+'/the_save_model.h5')
```

```
# Import models.
new_model = tf.keras.models.load_model(logdir+'/the_save_model.h5')
new_prediction = new_model.predict(test_x)
#np.testing.assert_allclose: determines whether the similarity of two objects exceeds the specified tolerance. If yes, the system displays an exception.
#atol: specified tolerance
np.testing.assert_allclose(result, new_prediction, atol=1e-6) # Prediction results are the same.
```

After a model is saved, you can find the corresponding weight file in the corresponding folder.

2.3.3.2 Saving and Loading Network Weights Only

If the weight name is suffixed with **.h5** or **.keras**, save the weight as an HDF5 file; otherwise, save the weight as a TensorFlow checkpoint file by default.

Code:

```
model.save_weights('./model/model_weights')
model.save_weights('./model/model_weights.h5')
# Load the weights.
model.load_weights('./model/model_weights')
model.load_weights('./model/model_weights.h5')
```

3 Handwritten Digit Recognition with TensorFlow

3.1 Introduction

Handwritten digit recognition is a common image recognition task where computers recognize digits in handwritten images. Different from printed fonts, handwriting of different people have different styles and sizes, making it difficult for computers to recognize handwriting. This exercise uses deep learning and TensorFlow tools to train a model using MNIST datasets.

This section describes the basic process of TensorFlow computing and basic elements for building a network.

3.2 Objectives

- Understand the basic TensorFlow calculation process;
- Be familiar with the basic elements for building a network, including dataset acquisition, network model building, model training, and model validation.

3.3 Tasks

- Read the MNIST handwritten digit dataset.
- Get started with TensorFlow by using simple mathematical models.
- Implement softmax regression by using high-level APIs.
- Build a multi-layer CNN.
- Implement a CNN by using high-level APIs.
- Visualize prediction results.

3.3.1 Project Description and Dataset Acquisition

3.3.1.1 Description

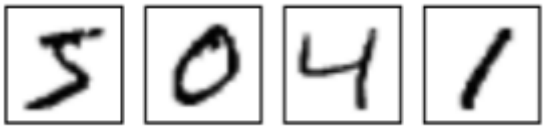
Handwritten digit recognition is a common image recognition task where computers recognize digits in handwritten images. Different from printed fonts, handwriting of different people have different styles and sizes, making it difficult for computers to

recognize handwriting. This exercise uses deep learning and TensorFlow tools to train a model using MNIST datasets.

3.3.1.2 Data Acquisition and Processing

3.3.1.2.1 About the Dataset

1. The MNIST dataset is provided by the National Institute of Standards and Technology (NIST).
2. It consists of handwritten digits from 250 different individuals, of which 50% are high school students and 50% are staff from Bureau of the Census.
3. You can download the dataset from http://yann.lecun.com/exdb/mnist/, which consists of the following parts:
   - Training set images: train-images-idx3-ubyte.gz (9.9 MB, 47 MB after decompression, including 60,000 samples)
   - Training set labels: train-labels-idx1-ubyte.gz (29 KB, 60 KB after decompression, including 60,000 labels)
   - Test set images: t10k-images-idx3-ubyte.gz (1.6 MB, 7.8 MB after decompression, including 10,000 samples)
   - Test set labels: t10k-labels-idx1-ubyte.gz (5 KB, 10 KB after decompression, including 10,000 labels)
4. The MNIST is an entry-level computer vision dataset that contains images of various handwritten digits.



It also contains one label corresponding to each image to clarify the correct digit. For example, the labels for the preceding four images are 5, 0, 4, and 1.

3.3.1.2.2 MNIST Dataset Reading

Download the MNIST dataset from the official TensorFlow website and decompress it.

Code:

```
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, optimizers, datasets
from matplotlib import pyplot as plt
import numpy as np

(x_train_raw, y_train_raw), (x_test_raw, y_test_raw) = datasets.mnist.load_data()

print(y_train_raw[0])
print(x_train_raw.shape, y_train_raw.shape)
print(x_test_raw.shape, y_test_raw.shape)

# Convert the labels into one-hot codes.
```

```
num_classes = 10
y_train = keras.utils.to_categorical(y_train_raw, num_classes)
y_test = keras.utils.to_categorical(y_test_raw, num_classes)
print(y_train[0])
```

Output:

```
5
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

In the MNIST dataset, the **images** is a tensor in the shape of [60000, 28, 28]. The first dimension is used to extract images, and the second and third dimensions are used to extract pixels in each image. Each element in this tensor indicates the strength of a pixel in an image. The value ranges from **0** to **255**.

The label data is converted from scalar to one-hot vectors. In a one-hot vector, one digit is 1 and digits in other dimensions are all 0s. For example, label 1 can be represented as [0,1,0,0,0,0,0,0,0,0]. Therefore, the labels are a digital matrix of [60000, 10].

3.3.2 Dataset Preprocessing and Visualization

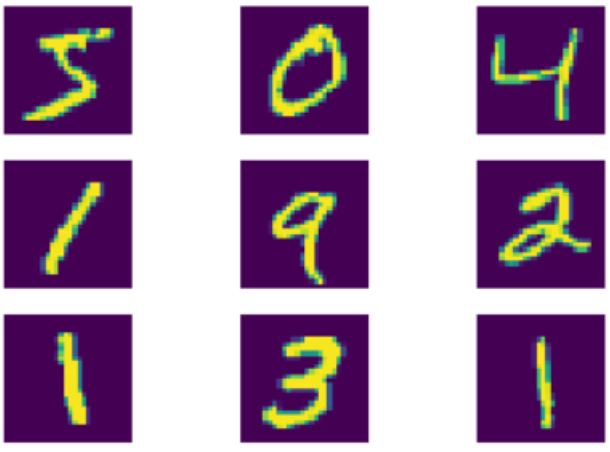
3.3.2.1 Data Visualization

Draw the first nine images.

Code:

```
plt.figure()
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(x_train_raw[i])
    #plt.ylabel(y[i].numpy())
    plt.axis('off')
plt.show()
```

Output:



Data processing: The output of a fully connected network must be in the form of vector, instead of the matrix form of the current images. Therefore, you need to sort the images into vectors.

Code:

```
# Convert a 28 x 28 image to a 784 x 1 vector.
x_train = x_train_raw.reshape(60000, 784)
x_test = x_test_raw.reshape(10000, 784)
```

Currently, the dynamic range of pixels is 0 to 255. Image pixels are usually normalized to the range of 0 to 1 during processing of image pixel values.

Code:

```
Code:
 # Normalize image pixel values.
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
```

3.3.3 DNN Construction

3.3.3.1 Building a DNN Model

Code:

```
# Create a deep neural network (DNN) model that consists of three fully connected layers and two ReLU activation functions.
model = keras.Sequential([
      layers.Dense(512, activation='relu', input_dim = 784),
      layers.Dense(256, activation='relu'),
      layers.Dense(124, activation='relu'),
layers.Dense(num_classes, activation='softmax')])

model.summary()
```

Output:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 124)               31868
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1250
=================================================================
Total params: 566,366
Trainable params: 566,366
Non-trainable params: 0
_________________________________________________________________
```

layer.Dense() indicates a fully connected layer, and **activation** indicates a used activation function.

3.3.3.2 Compiling the DNN Model

Code:

```
Optimizer = optimizers.Adam(0.001)
model.compile(loss=keras.losses.categorical_crossentropy,
                    optimizer=Optimizer,
                    metrics=['accuracy'])
```

In the preceding example, the loss function of the model is defined as cross entropy, and the optimization algorithm is the **Adam** gradient descent method.

3.3.3.3 Training the DNN Model

Code:

```
# Fit the training data to the model by using the fit method.
model.fit(x_train, y_train,
                batch_size=128,
                epochs=10,
                verbose=1)
```

Output:

```
Epoch 1/10
60000/60000 [==============================] - 7s 114us/sample - loss: 0.2281 - acc: 0.9327s - loss: 0.2594 - acc: 0. - ETA: 1s - loss: 0.2535 - acc: 0.9 - ETA: 1s - loss:
Epoch 2/10
60000/60000 [==============================] - 8s 129us/sample - loss: 0.0830 - acc: 0.9745s - loss: 0.0814 - ac
Epoch 3/10
60000/60000 [==============================] - 8s 127us/sample - loss: 0.0553 - acc: 0.9822
Epoch 4/10
60000/60000 [==============================] - 7s 117us/sample - loss: 0.0397 - acc: 0.9874s - los
Epoch 5/10
60000/60000 [==============================] - 8s 129us/sample - loss: 0.0286 - acc: 0.9914
Epoch 6/10
60000/60000 [==============================] - 8s 136us/sample - loss: 0.0252 - acc: 0.9919
Epoch 7/10
60000/60000 [==============================] - 8s 129us/sample - loss: 0.0204 - acc: 0.9931s - lo
Epoch 8/10
60000/60000 [==============================] - 8s 135us/sample - loss: 0.0194 - acc: 0.9938
Epoch 9/10
60000/60000 [==============================] - 7s 109us/sample - loss: 0.0162 - acc: 0.9948
Epoch 10/10
60000/60000 [==============================] - ETA: 0s - loss: 0.0149 - acc: 0.994 - 7s 117us/sample - loss: 0.0148 - acc: 0.9948
```

epoch indicates a specific round of training. In the preceding example, full data is iterated for 10 times.

3.3.3.4 Evaluating the DNN Model

Code:

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Output:

```
Test loss: 0.48341113169193267
Test accuracy: 0.8765
```

According to the evaluation, the model accuracy is 0.87 after 10 model training iterations.

3.3.3.5 Saving the DNN Model

Code:

```
logdir='./mnist_model'
if not os.path.exists(logdir):
    os.mkdir(logdir)
model.save(logdir+'/final_DNN_model.h5')
```

3.3.4 CNN Construction

The conventional CNN construction method helps you better understand the internal network structure but requires a large code volume. Therefore, attempts to construct a CNN by using high-level APIs are made to simplify the network construction process.

3.3.4.1 Building a CNN Model

Code:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

model=keras.Sequential() # Create a network sequence.
## Add the first convolutional layer and pooling layer.
model.add(keras.layers.Conv2D(filters=32,kernel_size = 5,strides = (1,1),
                              padding = 'same',activation = tf.nn.relu,input_shape = (28,28,1)))
model.add(keras.layers.MaxPool2D(pool_size=(2,2), strides = (2,2), padding = 'valid'))
## Add the second convolutional layer and pooling layer.
model.add(keras.layers.Conv2D(filters=64,kernel_size = 3,strides = (1,1),padding = 'same',activation =
tf.nn.relu))
model.add(keras.layers.MaxPool2D(pool_size=(2,2), strides = (2,2), padding = 'valid'))
## Add a dropout layer to reduce overfitting.
model.add(keras.layers.Dropout(0.25))
model.add(keras.layers.Flatten())
## Add two fully connected layers.
model.add(keras.layers.Dense(units=128,activation = tf.nn.relu))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(units=10,activation = tf.nn.softmax))
```

In the preceding network, two convolutional layers and pooling layers are first added by using **keras.layers**. Afterwards, a dropout layer is added to prevent overfitting. Finally, two full connection layers are added.

3.3.4.2 Compiling and Training the CNN Model

Code:

```
# Expand data dimensions to adapt to the CNN model.
X_train=x_train.reshape(60000,28,28,1)
X_test=x_test.reshape(10000,28,28,1)
model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=['accuracy'])
model.fit(x=X_train,y=y_train,epochs=5,batch_size=128)
```

Output:

```
Epoch 1/5
55000/55000 [==============================] - 49s 899us/sample - loss: 0.2107 - acc: 0.9348
Epoch 2/5
55000/55000 [==============================] - 48s 877us/sample - loss: 0.0793 - acc: 0.9763
Epoch 3/5
55000/55000 [==============================] - 52s 938us/sample - loss: 0.0617 - acc: 0.9815
Epoch 4/5
55000/55000 [==============================] - 48s 867us/sample - loss: 0.0501 - acc: 0.9846
Epoch 5/5
55000/55000 [==============================] - 50s 901us/sample - loss: 0.0452 - acc: 0.9862

<tensorflow.python.keras.callbacks.History at 0x214bbf34ac8>
```

During training, the network training data is iterated for only five times. You can increase the number of network iterations to check the effect.

3.3.4.3 Evaluating the CNN Model

Code:

```
test_loss,test_acc=model.evaluate(x=X_test,y=y_test)
print("Test Accuracy %.2f"%test_acc)
```

Output:

```
10000/10000 [==============================] - 2s 185us/sample - loss: 0.0239 - acc: 0.9921
Test Accuracy 0.99
```

The evaluation shows that the accuracy of the CNN model reaches up to 99%.

3.3.4.4 Saving the CNN Model

Code:

```
logdir='./mnist_model'
if not os.path.exists(logdir):
    os.mkdir(logdir)
model.save(logdir+'/final_CNN_model.h5')
```

Output:

```
10000/10000 [==============================] - 5s 489us/sample - loss: 0.0263 - acc: 0.9920s -
loss: 0.0273 - ac
Test Accuracy 0.99
```

3.3.5 Prediction Result Visualization

3.3.5.1 Loading the CNN Model

Code:

```
from tensorflow.keras.models import load_model
new_model = load_model('./mnist_model/final_CNN_model.h5')
new_model.summary()
```

Output:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout (Dropout)            (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               401536
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 422,154
Trainable params: 422,154
Non-trainable params: 0
_________________________________________________________________
```

Visualize the prediction results.

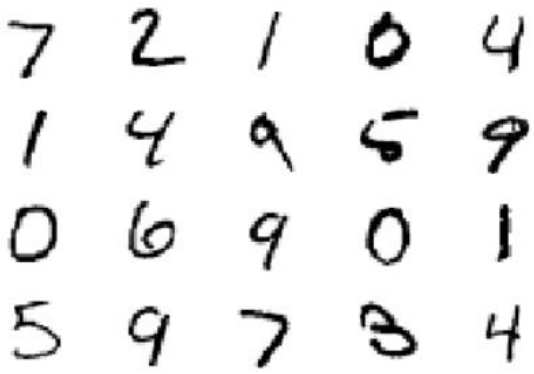
Code:

```
# Visualize the test dataset output.
import matplotlib.pyplot as plt
%matplotlib inline
def res_Visual(n):
    final_opt_a=new_model.predict_classes(X_test[0:n])# Perform predictions on the test dataset by
using the model.
    fig, ax = plt.subplots(nrows=int(n/5),ncols=5 )
```

```
    ax = ax.flatten()
    print('prediction results of the first {} images:'.format(n))
    for i in range(n):
        print(final_opt_a[i],end=',')
        if int((i+1)%5) ==0:
            print('\t')
      # Display images.
        img = X_test[i].reshape( (28,28))# Read each row of data in Ndarry format.
        plt.axis("off")
      ax[i].imshow(img, cmap='Greys', interpolation='nearest')# Visualization
        ax[i].axis("off")
    print('first {} images in the test dataset are in the following format:'.format(n))
res_Visual(20)
```

Output:

```
Prediction results of the first 20 images:
7,2,1,0,4,
1,4,9,5,9,
0,6,9,0,1,
5,9,7,3,4,
First 20 images in the test dataset:
```



4 Image Classification

4.1 Introduction

4.1.1 About This Exercise

This exercise is about a basic task in computer vision, that is, image recognition. The NumPy and TensorFlow modules are required. The NumPy module is used to create image objects, and the TensorFlow framework is used to create deep learning algorithms and build CNNs. This exercise recognizes image classes based on the CIFAR10 dataset.

4.1.2 Objectives

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

- Strengthen the understanding of the Keras-based neural network model building process.
- Master the method to load a pre-trained model.
- Learn how to use the checkpoint function.
- Learn how to use a trained model to make predictions.

4.2 Tasks

4.2.1 Importing Dependencies

```
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
```

4.2.2 Preprocessing Data

Code:

```
# Download the dataset.
(x_train,y_train), (x_test, y_test) = datasets.cifar10.load_data()
# Print the dataset size.
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
print(y_train[0])

# Convert the labels.
num_classes = 10
y_train_onehot = keras.utils.to_categorical(y_train, num_classes)
y_test_onehot = keras.utils.to_categorical(y_test, num_classes)
y_train[0]
```

Output:

```
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)
[6]

array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.], dtype=float32)
```

Display nine sample images.

Code:

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

Output:

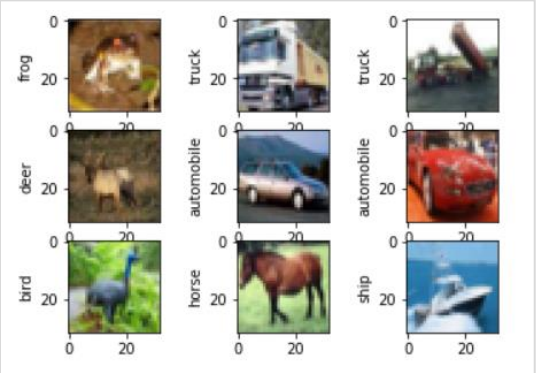


Figure 4-1 First nine images and their labels

Code:

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

4.2.3 Building a Model

Code:

```
def CNN_classification_model(input_size = x_train.shape[1:]):
    model = Sequential()
    # The first two modules have two convolutional layers and one pooling 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))
    model.add(Activation('relu'))
    #Conv2
    model.add(Conv2D(32, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2),strides =1))
    # The second module
    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

   # Perform flattening before connecting to the fully connected network.
    model.add(Flatten())
    model.add(Dense(128))
    model.add(Activation('relu'))
   # The dropout layer parameter value ranges from 0 to 1.
    model.add(Dropout(0.25))
    model.add(Dense(num_classes))
    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_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
activation_8 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
activation_9 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 29, 29, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 29, 29, 64)        18496
_________________________________________________________________
activation_10 (Activation)   (None, 29, 29, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 27, 27, 64)        36928
_________________________________________________________________
activation_11 (Activation)   (None, 27, 27, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 10816)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               1384576
_________________________________________________________________
activation_12 (Activation)   (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_13 (Activation)   (None, 10)                0
=================================================================
Total params: 1,451,434
Trainable params: 1,451,434
Non-trainable params: 0
_________________________________________________________________
```

4.2.4 Training the Model

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 the pre-trained model.
trained_weights_path = 'cifar10_weights.h5'
if os.path.exists(trained_weights_path):
    model.load_weights(trained_weights_path, by_name =True)
# Training
model.fit(x_train,y_train, batch_size=32, epochs=10,callbacks = [model_checkpoint],verbose=1)
```

Output:

```
Train on 50000 samples
Epoch 1/10
49984/50000 [============================>.] - ETA: 0s - loss: 1.6541 - accuracy: 0.3961
Epoch 00001: loss improved from inf to 1.65395, saving model to final_cifar10.h5
50000/50000 [==============================] - 322s 6ms/sample - loss: 1.6540 - accuracy: 0.3961
Epoch 2/10
49984/50000 [============================>.] - ETA: 0s - loss: 1.3494 - accuracy: 0.5171
Epoch 00002: loss improved from 1.65395 to 1.34929, saving model to final_cifar10.h5
50000/50000 [==============================] - 284s 6ms/sample - loss: 1.3493 - accuracy: 0.5171
Epoch 3/10
49984/50000 [============================>.] - ETA: 0s - loss: 1.2141 - accuracy: 0.5709
Epoch 00003: loss improved from 1.34929 to 1.21421, saving model to final_cifar10.h5
50000/50000 [==============================] - 303s 6ms/sample - loss: 1.2142 - accuracy: 0.5709
Epoch 4/10
49984/50000 [============================>.] - ETA: 0s - loss: 1.1104 - accuracy: 0.6113
Epoch 00004: loss improved from 1.21421 to 1.11042, saving model to final_cifar10.h5
50000/50000 [==============================] - 291s 6ms/sample - loss: 1.1104 - accuracy: 0.6112
Epoch 5/10
49984/50000 [============================>.] - ETA: 0s - loss: 1.0166 - accuracy: 0.6444
Epoch 00005: loss improved from 1.11042 to 1.01649, saving model to final_cifar10.h5
50000/50000 [==============================] - 293s 6ms/sample - loss: 1.0165 - accuracy: 0.6445
Epoch 6/10
49984/50000 [============================>.] - ETA: 0s - loss: 0.9419 - accuracy: 0.6715
Epoch 00006: loss improved from 1.01649 to 0.94189, saving model to final_cifar10.h5
50000/50000 [==============================] - 281s 6ms/sample - loss: 0.9419 - accuracy: 0.6715
Epoch 7/10
49984/50000 [============================>.] - ETA: 0s - loss: 0.8931 - accuracy: 0.6873
Epoch 00007: loss improved from 0.94189 to 0.89316, saving model to final_cifar10.h5
50000/50000 [==============================] - 280s 6ms/sample - loss: 0.8932 - accuracy: 0.6872
Epoch 8/10
49984/50000 [============================>.] - ETA: 0s - loss: 0.8367 - accuracy: 0.7095
Epoch 00008: loss improved from 0.89316 to 0.83656, saving model to final_cifar10.h5
50000/50000 [==============================] - 306s 6ms/sample - loss: 0.8366 - accuracy: 0.7095
Epoch 9/10
49984/50000 [============================>.] - ETA: 0s - loss: 0.7928 - accuracy: 0.7238
Epoch 00009: loss improved from 0.83656 to 0.79273, saving model to final_cifar10.h5
50000/50000 [==============================] - 304s 6ms/sample - loss: 0.7927 - accuracy: 0.7238
Epoch 10/10
49984/50000 [============================>.] - ETA: 0s - loss: 0.7423 - accuracy: 0.7401
Epoch 00010: loss improved from 0.79273 to 0.74234, saving model to final_cifar10.h5
50000/50000 [==============================] - 286s 6ms/sample - loss: 0.7423 - accuracy: 0.7401

<tensorflow.python.keras.callbacks.History at 0x18608947908>
```

This exercise is performed on a laptop. The network in this exercise 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.

4.2.5 Evaluating the Model

Code:

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

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

Output:

```
10000/10000 [==============================] - 13s 1ms/sample - loss: 0.8581 - accuracy:
0.7042s - loss: 0.854
[0.8581173644065857, 0.7042]
```

Predict an 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 prediction result.
new_model.predict_classes(x_test[0:1])
```

Output:

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

Output the first four images and their prediction results.

Code:

```
pred_list = []

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

Output:

Figure 4-2 Images and prediction results

4.3 Summary

This section describes how to build an image classification model based on TensorFlow 2 and Python. It provides trainees with basic concepts in building deep learning models.

Huawei AI Certification Training

HCIA-AI

Huawei AI Computing Framework

MindSpore

Lab Guide

ISSUE: 3.0

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

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

Notice

The purchased products, services, and features are stipulated by the commercial contract made between Huawei and the customer. All or partial products, services, and features described in this document may not be within the purchased scope or the usage scope. Unless otherwise agreed by 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 the 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:	https://e.huawei.com/en/

Huawei Certification System

Huawei Certification is an integral part of the company's "Platform + Ecosystem" strategy, it supports the ICT infrastructure featuring "Cloud-Pipe-Device". It evolves to reflect the latest trends of ICT development. Huawei Certification consists of two categories: ICT Infrastructure, and Cloud Service & Platform.

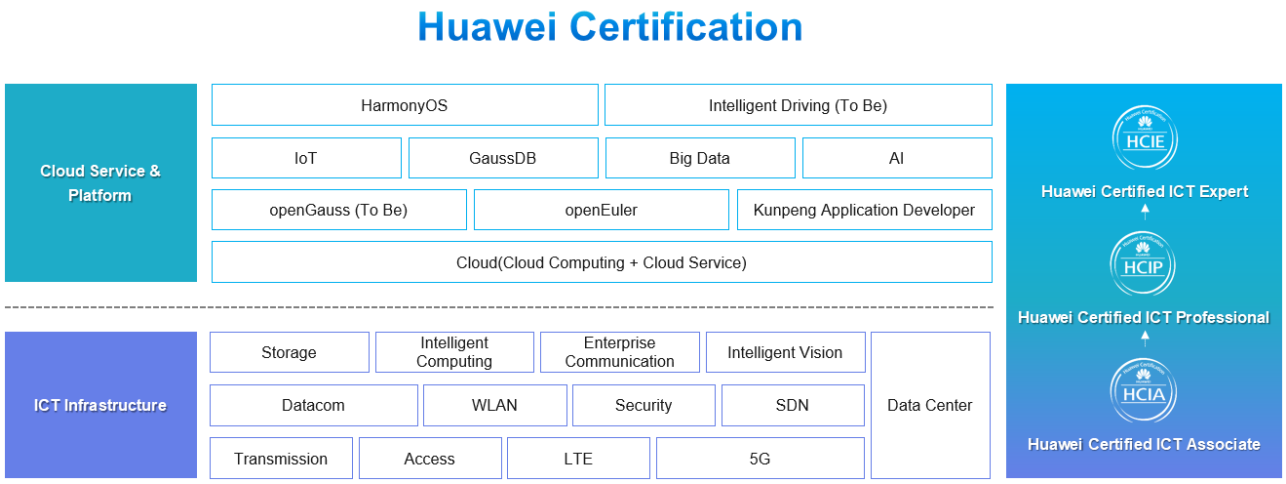
Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

With its leading talent development system and certification standards, Huawei is committed to developing ICT professionals in the digital era, building a healthy ICT talent ecosystem.

HCIA-AI v3.0 certification is intended for cultivating and conducting qualification of engineers who are capable of creatively designing and developing AI products and solutions using machine learning and deep learning algorithms.

An HCIA-AI v3.0 certificate proves that you: Have understood the development history of AI, Huawei Ascend AI system, and Huawei full-stack AI strategy in all scenarios. Have mastered traditional machine learning and deep learning Are able to use the TensorFlow and MindSpore frameworks to build, train, and deploy neural networks. Competent in sales, marketing, product manager, project management, and technical support positions in the AI field.

Huawei Certification Portfolio



About This Document

Overview

This document is applicable to the candidates who are preparing for the HCIA-AI exam and readers who want to understand basic AI knowledge and basic MindSpore programming.

Description

This lab guide consists of the following two exercises:

- Exercise 1: MindSpore basics, which describes the basic syntax and common modules of MindSpore.
- Exercise 2: Handwritten character recognition, in which the MindSpore framework is used to recognize handwritten characters.

Background Knowledge Required

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

- Have basic Python knowledge, be familiar with basic MindSpore concepts, and understand basic Python programming knowledge.

Contents

1 MindSpore Basics

1.1 Introduction

1.1.1 About This Exersie

This exercise introduces the tensor data structure of MindSpore. By performing a series of operations on tensors, you can understand the basic syntax of MindSpore.

1.1.2 Objectives

- Master the method of creating tensors.
- Master the properties and methods of tensors.

1.1.3 Lab Environment

MindSpore 1.2 or later is recommended. The exercise can be performed on a PC or by logging in to HUAWEI CLOUD and purchasing the ModelArts service.

1.2 Procedure

1.2.1 Introduction to Tensors

Tensor is a basic data structure in the MindSpore network computing. For details about data types in tensors, see dtype.

Tensors of different dimensions represent different data. For example, a 0-dimensional tensor represents a scalar, a 1-dimensional tensor represents a vector, a 2-dimensional tensor represents a matrix, and a 3-dimensional tensor may represent the three channels of RGB images.

MindSpore tensors support different data types, including int8, int16, int32, int64, uint8, uint16, uint32, uint64, float16, float32, float64 and bool, which correspond to the data types of NumPy.

In the computation process of MindSpore, the int data type in Python is converted into the defined int64 type, and the float data type is converted into the defined float32 type.

1.2.1.1 Creating a Tensor

During tensor construction, the tensor, float, int, Boolean, tuple, list, and NumPy.array types can be input. The tuple and list can store only data of the float, int, and Boolean types.

The data type can be specified during tensor initialization. However, if the data type is not specified, the initial **int**, **float**, and **bool** values respectively generate 0-dimensional tensors with mindspore.int32, mindspore.float32 and mindspore.bool_ data types. The data types of the 1-dimensional tensors generated by the initial values **tuple** and **list** correspond to those of tensors stored in the tuple and list. If multiple types of data are contained, the MindSpore data type corresponding to the type with the highest priority is selected (Boolean < int < float). If the initial value is **Tensor**, the data type is tensor. If the initial value is NumPy.array, the generated tensor data type corresponds to NumPy.array.

Step 1 Create a tensor using an array.

Code:

```
# Import MindSpore.
import mindspore
# The cell outputs multiple lines at the same time.
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import numpy as np
from mindspore import Tensor
from mindspore import dtype
# Use an array to create a tensor.
x = Tensor(np.array([[1, 2], [3, 4]]), dtype.int32)
x
```

Output:

```
Tensor(shape=[2, 2], dtype=Int32, value=
[[1, 2],
 [3, 4]])
```

Step 2 Create tensors using numbers.

Code:

```
# Use numbers to create tensors.
y = Tensor(1.0, dtype.int32)
z = Tensor(2, dtype.int32)
y
z
```

Output:

```
Tensor(shape=[], dtype=Int32, value= 1)
Tensor(shape=[], dtype=Int32, value= 2)
```

Step 3 Create a tensor using Boolean.

Code:

```
# Use Boolean to create a tensor.
m = Tensor(True, dtype.bool_)
m
```

Output:

```
Tensor(shape=[], dtype=Bool, value= True)
```

Step 4 Create a tensor using a tuple.

Code:

```
# Use a tuple to create a tensor.
n = Tensor((1, 2, 3), dtype.int16)
n
```

Output:

```
Tensor(shape=[3], dtype=Int16, value= [1, 2, 3])
```

Step 5 Create a tensor using a list.

Code:

```
# Use a list to create a tensor.
p = Tensor([4.0, 5.0, 6.0], dtype.float64)
p
```

Output:

```
Tensor(shape=[3], dtype=Float64, value= [4.00000000e+000, 5.00000000e+000, 6.00000000e+000]
```

Step 6 Inherit attributes of another tensor to form a new tensor.

Code:

```
from mindspore import ops
oneslike = ops.OnesLike()
x = Tensor(np.array([[0, 1], [2, 1]]).astype(np.int32))
output = oneslike(x)
output
```

Output:

```
Tensor(shape=[2, 2], dtype=Int32, value=
[[1, 1],
 [1, 1]])
```

Step 7 Output constant tensor value.

Code:

```
from mindspore.ops import operations as ops

shape = (2, 2)
ones = ops.Ones()
output = ones(shape,dtype.float32)
print(output)

zeros = ops.Zeros()
output = zeros(shape, dtype.float32)
print(output)
```

Output:

```
[[1. 1.]
 [1. 1.]]
[[0. 0.]
 [0. 0.]]
```

1.2.1.2 Tensor Attributes

Tensor attributes include shape and data type (dtype).

- Shape: a tuple
- Dtype: a data type of MindSpore

Code:

```
x = Tensor(np.array([[1, 2], [3, 4]]), dtype.int32)

x.shape # Shape
x.dtype # Data type
x.ndim   # Dimension
x.size # Size
```

Output:

```
(2, 2)
mindspore.int32
2
4
```

1.2.1.3 Tensor Methods

asnumpy(): converts a tensor to an array of NumPy.

Code:

```
y = Tensor(np.array([[True, True], [False, False]]), dtype.bool_)

# Convert the tensor data type to NumPy.
y_array = y.asnumpy()

y
y_array
```

Output:

```
Tensor(shape=[2, 2], dtype=Bool, value=
[[ True,  True],
 [False, False]])

array([[ True,  True],
       [False, False]])
```

1.2.1.4 Tensor Operations

There are many operations between tensors, including arithmetic, linear algebra, matrix processing (transposing, indexing, and slicing), and sampling. The following describes several operations. The usage of tensor computation is similar to that of NumPy.

Step 1 Perform indexing and slicing.

Code:

```
tensor = Tensor(np.array([[0, 1], [2, 3]]).astype(np.float32))
print("First row: {}".format(tensor[0]))
print("First column: {}".format(tensor[:, 0]))
print("Last column: {}".format(tensor[..., -1]))
```

Output:

```
First row: [0. 1.]
First column: [0. 2.]
Last column: [1. 3.]
```

Step 2 Concatenate tensors.

Code:

```
data1 = Tensor(np.array([[0, 1], [2, 3]]).astype(np.float32))
data2 = Tensor(np.array([[4, 5], [6, 7]]).astype(np.float32))
op = ops.Stack()
output = op([data1, data2])
print(output)
```

Output:

```
[[[0. 1.]
  [2. 3.]]

 [[4. 5.]
  [6. 7.]]]
```

Step 3 Convert to NumPy.

Code:

```
zeros = ops.Zeros()
output = zeros((2,2), dtype.float32)
print("output: {}".format(type(output)))
n_output = output.asnumpy()
print("n_output: {}".format(type(n_output)))
```

Output:

```
output: <class 'mindspore.common.tensor.Tensor'>
n_output: <class 'numpy.ndarray'>
```

1.2.2 Loading a Dataset

MindSpore.dataset provides APIs to load and process datasets, such as MNIST, CIFAR-10, CIFAR-100, VOC, ImageNet, and CelebA.

Step 1 Load the MNIST dataset.

You are advised to download the MNIST dataset from http://yann.lecun.com/exdb/mnist/ and save the training and test files to the MNIST folder.

Code:

```
import os
import mindspore.dataset as ds
import matplotlib.pyplot as plt

dataset_dir = "./MNIST/train"    # Path of the dataset
# Read three images from the MNIST dataset.
mnist_dataset = ds.MnistDataset(dataset_dir=dataset_dir, num_samples=3)
# View the images and set the image sizes.
plt.figure(figsize=(8,8))
i = 1
# Print three subgraphs.
for dic in mnist_dataset.create_dict_iterator():
    plt.subplot(3,3,i)
    plt.imshow(dic['image'].asnumpy())
    plt.axis('off')
    i +=1
plt.show()
```

Output:

图1-1 MNIST dataset sample

Step 2 Customize a dataset.

For datasets that cannot be directly loaded by MindSpore, you can build a custom dataset class and use the **GeneratorDataset** API to customize data loading.

Code:

```
import numpy as np
np.random.seed(58)

class DatasetGenerator:
# When a dataset object is instantiated, the __init__ function is called. You can perform operations such as data initialization.
    def __init__(self):
        self.data = np.random.sample((5, 2))
        self.label = np.random.sample((5, 1))
# Define the __getitem__ function of the dataset class to support random access and obtain and return data in the dataset based on the specified index value.
    def __getitem__(self, index):
        return self.data[index], self.label[index]
# Define the __len__ function of the dataset class and return the number of samples in the dataset.
    def __len__(self):
        return len(self.data)
# After the dataset class is defined, the GeneratorDataset API can be used to load and access dataset samples in the user-defined mode.
dataset_generator = DatasetGenerator()
dataset = ds.GeneratorDataset(dataset_generator, ["data", "label"], shuffle=False)
# Use the create_dict_iterator method to obtain data.
for data in dataset.create_dict_iterator():
    print('{}'.format(data["data"]), '{}'.format(data["label"]))
```

Output:

```
[0.36510558 0.45120592] [0.78888122]
[0.49606035 0.07562207] [0.38068183]
[0.57176158 0.28963401] [0.16271622]
[0.30880446 0.37487617] [0.54738768]
[0.81585667 0.96883469] [0.77994068]
```

Step 3 Perform data augmentation.

The dataset APIs provided by MindSpore support data processing methods, such as shuffle and batch. You only need to call the corresponding function API to quickly process data.

In the following example, the datasets are shuffled, and then two samples form a batch.

Code:

```
ds.config.set_seed(58)

# Shuffle the data sequence. buffer_size indicates the size of the shuffled buffer in the dataset.
dataset = dataset.shuffle(buffer_size=10)
# Divide the dataset into batches. batch_size indicates the number of data records contained in each batch. Set this parameter to 2.
dataset = dataset.batch(batch_size=2)

for data in dataset.create_dict_iterator():
    print("data: {}".format(data["data"]))
    print("label: {}".format(data["label"]))
```

Output:

```
data: [[0.36510558 0.45120592]
 [0.57176158 0.28963401]]
label: [[0.78888122]
 [0.16271622]]
data: [[0.30880446 0.37487617]
 [0.49606035 0.07562207]]
label: [[0.54738768]
 [0.38068183]]
data: [[0.81585667 0.96883469]]
label: [[0.77994068]]
```

Code:

```
import matplotlib.pyplot as plt

from mindspore.dataset.vision import Inter
import mindspore.dataset.vision.c_transforms as c_vision

DATA_DIR = './MNIST/train'
# Obtain six samples.
mnist_dataset = ds.MnistDataset(DATA_DIR, num_samples=6, shuffle=False)
# View the original image data.
mnist_it = mnist_dataset.create_dict_iterator()
data = next(mnist_it)
plt.imshow(data['image'].asnumpy().squeeze(), cmap=plt.cm.gray)
plt.title(data['label'].asnumpy(), fontsize=20)
plt.show()
```

Output:

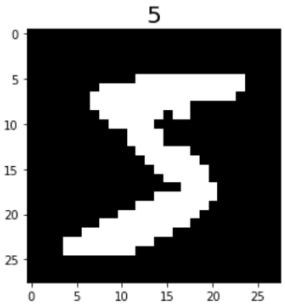


图1-2 Data sample

Code:

```
resize_op = c_vision.Resize(size=(40,40), interpolation=Inter.LINEAR)
crop_op = c_vision.RandomCrop(28)
transforms_list = [resize_op, crop_op]
mnist_dataset = mnist_dataset.map(operations=transforms_list, input_columns=["image"])
mnist_dataset = mnist_dataset.create_dict_iterator()
data = next(mnist_dataset)
plt.imshow(data['image'].asnumpy().squeeze(), cmap=plt.cm.gray)
plt.title(data['label'].asnumpy(), fontsize=20)
plt.show()
```

Output:

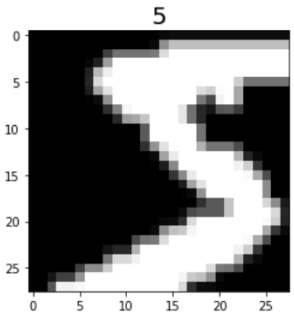


图1-3 Effect after data argumentation

1.2.3 Building the Network

MindSpore encapsulates APIs for building network layers in the nn module. Different types of neural network layers are built by calling these APIs.

Step 1 Build a fully-connected layer.

Fully-connected layer: mindspore.nn.Dense

- **in_channels**: input channel
- **out_channels**: output channel
- **weight_init**: weight initialization. Default: 'normal'.

Code:

```
import mindspore as ms
import mindspore.nn as nn
from mindspore import Tensor
import numpy as np

# Construct the input tensor.
input_a = Tensor(np.array([[1, 1, 1], [2, 2, 2]]), ms.float32)
print(input_a)
# Construct a fully-connected network. Set both in_channels and out_channels to 3.
net = nn.Dense(in_channels=3, out_channels=3, weight_init=1)
output = net(input_a)
print(output)
```

Output:

```
[[1. 1. 1.]
  [2. 2. 2.]]
[[3. 3. 3.]
  [6. 6. 6.]]
```

Step 2 Build a convolutional layer.

Code:

```
conv2d = nn.Conv2d(1, 6, 5, has_bias=False, weight_init='normal', pad_mode='valid')
input_x = Tensor(np.ones([1, 1, 32, 32]), ms.float32)

print(conv2d(input_x).shape)
```

Output:

```
(1, 6, 28, 28)
```

Step 3 Build a ReLU layer.

Code:

```
relu = nn.ReLU()
input_x = Tensor(np.array([-1, 2, -3, 2, -1]), ms.float16)
output = relu(input_x)

print(output)
```

Output:

```
[0. 2. 0. 2. 0.]
```

Step 4 Build a pooling layer.

Code:

```
max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
input_x = Tensor(np.ones([1, 6, 28, 28]), ms.float32)

print(max_pool2d(input_x).shape)
```

Output:

```
(1, 6, 14, 14)
```

Step 5 Build a flatten layer.

Code:

```
flatten = nn.Flatten()
input_x = Tensor(np.ones([1, 16, 5, 5]), ms.float32)
output = flatten(input_x)

print(output.shape)
```

Output:

```
(1, 400)
```

Step 6 Define a model class and view parameters.

The **Cell** class of MindSpore is the base class for building all networks and the basic unit of a network. When a neural network is required, you need to inherit the **Cell** class and overwrite the **__init__** and **construct** methods.

Code:

```
class LeNet5(nn.Cell):
    """
    LeNet network structure
    """
    def __init__(self, num_class=10, num_channel=1):
        super(LeNet5, self).__init__()
        # Define the required operation.
        self.conv1 = nn.Conv2d(num_channel, 6, 5, pad_mode='valid')
        self.conv2 = nn.Conv2d(6, 16, 5, pad_mode='valid')
        self.fc1 = nn.Dense(16 * 4 * 4, 120)
        self.fc2 = nn.Dense(120, 84)
        self.fc3 = nn.Dense(84, num_class)
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()

    def construct(self, x):
        # Use the defined operation to build a forward network.
        x = self.conv1(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x
# Instantiate the model and use the parameters_and_names method to view the model parameters.
modelle = LeNet5()
for m in modelle.parameters_and_names():
    print(m)
```

Output:

```
('conv1.weight', Parameter (name=conv1.weight, shape=(6, 1, 5, 5), dtype=Float32, requires_grad=True))
('conv2.weight', Parameter (name=conv2.weight, shape=(16, 6, 5, 5), dtype=Float32,
requires_grad=True))
('fc1.weight', Parameter (name=fc1.weight, shape=(120, 400), dtype=Float32, requires_grad=True))
('fc1.bias', Parameter (name=fc1.bias, shape=(120,), dtype=Float32, requires_grad=True))
('fc2.weight', Parameter (name=fc2.weight, shape=(84, 120), dtype=Float32, requires_grad=True))
('fc2.bias', Parameter (name=fc2.bias, shape=(84,), dtype=Float32, requires_grad=True))
('fc3.weight', Parameter (name=fc3.weight, shape=(10, 84), dtype=Float32, requires_grad=True))
('fc3.bias', Parameter (name=fc3.bias, shape=(10,), dtype=Float32, requires_grad=True))
```

1.2.4 Training and Validating a Model

Step 1 Use loss functions.

A loss function is used to validate the difference between the predicted and actual values of a model. Here, the absolute error loss function **L1Loss** is used. **mindspore.nn.loss** also provides many other loss functions, such as **SoftmaxCrossEntropyWithLogits**, **MSELoss**, and **SmoothL1Loss**.

The output value and target value are provided to compute the loss value. The method is as follows:

Code:

```
import numpy as np
import mindspore.nn as nn
from mindspore import Tensor
import mindspore.dataset as ds
import mindspore as ms
loss = nn.L1Loss()
output_data = Tensor(np.array([[1, 2, 3], [2, 3, 4]]).astype(np.float32))
target_data = Tensor(np.array([[0, 2, 5], [3, 1, 1]]).astype(np.float32))
print(loss(output_data, target_data))
```

Output:

```
1.5
```

Step 2 Use an optimizer.

Common deep learning optimization algorithms include **SGD**, **Adam**, **Ftrl**, **lazyadam**, **Momentum**, **RMSprop**, **Lars**, **Proximal_ada_grad**, and **lamb**.

Momentum optimizer: mindspore.nn.Momentum

Code:

```
optim = nn.Momentum(params=modelle.trainable_params(), learning_rate=0.1, momentum=0.9,
weight_decay=0.0)
```

Step 3 Build a model.

mindspore.Model(network, loss_fn, optimizer, metrics)

Code:

```
from mindspore import Model

# Define a neural network.
net = LeNet5()
# Define the loss function.
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
# Define the optimizer.
optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
# Build a model.
model = Model(network = net, loss_fn=loss, optimizer=optim, metrics={'accuracy'})
```

Step 4 Train the model.

Code:

```
import mindspore.dataset.transforms.c_transforms as C
import mindspore.dataset.vision.c_transforms as CV
from mindspore.train.callback import    LossMonitor

DATA_DIR = './MNIST/train'
mnist_dataset = ds.MnistDataset(DATA_DIR)

resize_op = CV.Resize((28,28))
rescale_op = CV.Rescale(1/255,0)
hwc2chw_op = CV.HWC2CHW()

mnist_dataset    = mnist_dataset .map(input_columns="image", operations=[rescale_op,resize_op,
hwc2chw_op])
mnist_dataset    = mnist_dataset .map(input_columns="label", operations=C.TypeCast(ms.int32))
mnist_dataset = mnist_dataset.batch(32)
loss_cb = LossMonitor(per_print_times=1000)
# dataset is an input parameter, which indicates the training set, and epoch indicates the number of
training epochs of the training set.
model.train(epoch=1, train_dataset=mnist_dataset,callbacks=[loss_cb])
```

Step 5 Validate the model.

Code:

```
# dataset is an input parameter, which indicates the validation set.
DATA_DIR = './forward_mnist/MNIST/test'
dataset = ds.MnistDataset(DATA_DIR)

resize_op = CV.Resize((28,28))
rescale_op = CV.Rescale(1/255,0)
hwc2chw_op = CV.HWC2CHW()

dataset    = dataset .map(input_columns="image", operations=[rescale_op,resize_op, hwc2chw_op])
dataset    = dataset .map(input_columns="label", operations=C.TypeCast(ms.int32))
dataset = dataset.batch(32)
model.eval(valid_dataset=dataset)
```

1.2.5 Auto Differentiation

Backward propagation is the commonly used algorithm for training neural networks. In this algorithm, parameters (model weights) are adjusted based on a gradient of a loss function for a given parameter.

The first-order derivative method of MindSpore is **mindspore.ops.GradOperation (get_all=False, get_by_list=False, sens_param=False)**. When **get_all** is set to **False**, the first input derivative is computed. When **get_all** is set to **True**, all input derivatives are computed. When **get_by_list** is set to **False**, weight derivatives are not computed. When **get_by_list** is set to **True**, weight derivatives are computed. **sens_param** scales the output value of the network to change the final gradient.

The following uses the MatMul operator derivative for in-depth analysis.

Step 1 Compute the first-order derivative of the input.

To compute the input derivative, you need to define a network requiring a derivative. The following uses a network **f(x,y)=z*x*y** formed by the MatMul operator as an example.

Code:

```
import numpy as np
import mindspore.nn as nn
import mindspore.ops as ops
from mindspore import Tensor
from mindspore import ParameterTuple, Parameter
from mindspore import dtype as mstype

class Net(nn.Cell):
    def __init__(self):
        super(Net, self).__init__()
        self.matmul = ops.MatMul()
        self.z = Parameter(Tensor(np.array([1.0], np.float32)), name='z')

    def construct(self, x, y):
        x = x * self.z
        out = self.matmul(x, y)
        return out
class GradNetWrtX(nn.Cell):
    def __init__(self, net):
        super(GradNetWrtX, self).__init__()
        self.net = net
        self.grad_op = ops.GradOperation()

    def construct(self, x, y):
        gradient_function = self.grad_op(self.net)
        return gradient_function(x, y)
x = Tensor([[0.8, 0.6, 0.2], [1.8, 1.3, 1.1]], dtype=mstype.float32)
y = Tensor([[0.11, 3.3, 1.1], [1.1, 0.2, 1.4], [1.1, 2.2, 0.3]], dtype=mstype.float32)
output = GradNetWrtX(Net())(x, y)
print(output)
```

Output:

```
[[4.5099998 2.7         3.6000001]
 [4.5099998 2.7         3.6000001]]
```

Step 2 Compute the first-order derivative of the weight.

To compute weight derivatives, you need to set **get_by_list** in **ops.GradOperation** to **True**. If computation of certain weight derivatives is not required, set **requirements_grad** to **False** when defining the network requiring derivatives.

Code:

```
class GradNetWrtX(nn.Cell):
    def __init__(self, net):
        super(GradNetWrtX, self).__init__()
        self.net = net
        self.params = ParameterTuple(net.trainable_params())
        self.grad_op = ops.GradOperation(get_by_list=True)

    def construct(self, x, y):
        gradient_function = self.grad_op(self.net, self.params)
        return gradient_function(x, y)
output = GradNetWrtX(Net())(x, y)
print(output)
```

Output:

```
(Tensor(shape=[1], dtype=Float32, value= [ 2.15359993e+01]),)
```

2 MNIST Handwritten Character Exercise

2.1 Introduction

This exercise implements the MNIST handwritten character recognition, which is a typical case in the deep learning field. The whole process is as follows:

- Process the required dataset. The MNIST dataset is used in this example.
- Define a network. A simple fully-connected network is built in this example.
- Define a loss function and an optimizer.
- Load the dataset and perform training. After the training is complete, use the test set for validation.

2.2 Preparations

Before you start, check whether MindSpore has been correctly installed. You are advised to install MindSpore 1.1.1 or later on your computer by referring to the MindSpore official website https://www.mindspore.cn/install/.

In addition, you shall have basic mathematical knowledge such as Python coding basics, probability, and matrix.

2.3 Detailed Design and Implementation

2.3.1 Data Preparation

The MNIST dataset used in this example consists of 10 classes of 28 x 28 pixels grayscale images. It has a training set of 60,000 examples, and a test set of 10,000 examples.

Download the MNIST dataset at http://yann.lecun.com/exdb/mnist/. Four dataset download links are provided. The first two links are for downloading test data files, and the last two links are for downloading training data files.

Download and decompress the files, and store them in the workspace directories **./MNIST /train** and **./MNIST /test**.

The directory structure is as follows:

```
└─MNIST
```

```
├─ test
│      t10k-images.idx3-ubyte
│      t10k-labels.idx1-ubyte
│
└─ train
       train-images.idx3-ubyte
       train-labels.idx1-ubyte
```

2.3.2 Procedure

Step 1 Import the Python library and module and configure running information.

Import the required Python library.

Currently, the **os** library is required. Other required libraries will not be described here. For details about the MindSpore modules, see the MindSpore API page. You can use **context.set_context** to configure the information required for running, such as the running mode, backend information, and hardware information.

Import the **context** module and configure the required information.

Code:

```
# Import related dependent libraries.
import   os
from matplotlib import pyplot as plt
import numpy as np

import mindspore as ms
import mindspore.context as context
import mindspore.dataset as ds
import mindspore.dataset.transforms.c_transforms as C
import mindspore.dataset.vision.c_transforms as CV
from mindspore.nn.metrics import Accuracy

from mindspore import nn
from mindspore.train import Model
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor

context.set_context(mode=context.GRAPH_MODE, device_target='CPU')
```

The graph mode is used in this exercise. You can configure hardware information as required. For example, if the code runs on the Ascend AI processor, set **device_target** to **Ascend**. This rule also applies to the code running on the CPU and GPU. For details about parameters, see the API description of **context.set_context**.

Step 2 Read data.

Use the data reading function of MindSpore to read the MNIST dataset and view the data volume and sample information of the training set and test set.

Code:

```
DATA_DIR_TRAIN = "MNIST/train" # Training set information
DATA_DIR_TEST = "MNIST/test" # Test set information
# Read data.
ds_train = ds.MnistDataset(DATA_DIR_TRAIN)
ds_test = ds.MnistDataset(DATA_DIR_TEST )
# Display the dataset features.
print('Data volume of the training dataset:',ds_train.get_dataset_size())
print(' Data volume of the test dataset:',ds_test.get_dataset_size())
image=ds_train.create_dict_iterator().__next__()
print('Image length/width/channels:',image['image'].shape)
print('Image label style:',image['label'])      # Total 10 label classes which are represented by numbers from 0 to 9.
```

Step 3 Process data.

Datasets are crucial for training. A good dataset can effectively improve training accuracy and efficiency. Generally, before loading a dataset, you need to perform some operations on the dataset.

Define a dataset and data operations.

We define the **create_dataset** function to create a dataset. In this function, we define the data augmentation and processing operations to be performed:

- Read the dataset.
- Define parameters required for data augmentation and processing.
- Generate corresponding data augmentation operations according to the parameters.
- Use the **map** function to apply data operations to the dataset.
- Process the generated dataset.

Code:

```
def create_dataset(training=True, batch_size=128, resize=(28, 28),
                   rescale=1/255, shift=0, buffer_size=64):
    ds = ms.dataset.MnistDataset(DATA_DIR_TRAIN if training else DATA_DIR_TEST)
    # Define the resizing, normalization, and channel conversion of the map operation.
    resize_op = CV.Resize(resize)
    rescale_op = CV.Rescale(rescale,shift)
    hwc2chw_op = CV.HWC2CHW()
    # Perform the map operation on the dataset.
    ds = ds.map(input_columns="image", operations=[rescale_op,resize_op, hwc2chw_op])
    ds = ds.map(input_columns="label", operations=C.TypeCast(ms.int32))
    # Set the shuffle parameter and batch size.
    ds = ds.shuffle(buffer_size=buffer_size)
    ds = ds.batch(batch_size, drop_remainder=True)
    return ds
```

In the preceding information, **batch_size** indicates the number of data records in each batch. Assume that each batch contains 32 data records. Modify the image size, normalization, and image channel, and then modify the data type of the label. Perform the shuffle operation, set **batch_size**, and set **drop_remainder** to **True**. In this case, data that cannot form a batch in the dataset will be discarded.

MindSpore supports multiple data processing and argumentation operations, which are usually used together. For details, see Data Processing and Data Argumentation.

Step 4 Visualize samples.

Read the first 10 samples and visualize the samples to determine whether the samples are real datasets.

Code:

```
# Display the first 10 images and the labels, and check whether the images are correctly labeled.
ds = create_dataset(training=False)
data = ds.create_dict_iterator().__next__()
images = data['image'].asnumpy()
labels = data['label'].asnumpy()
plt.figure(figsize=(15,5))
for i in range(1,11):
    plt.subplot(2, 5, i)
    plt.imshow(np.squeeze(images[i]))
    plt.title('Number: %s' % labels[i])
    plt.xticks([])
plt.show()
```

Output:

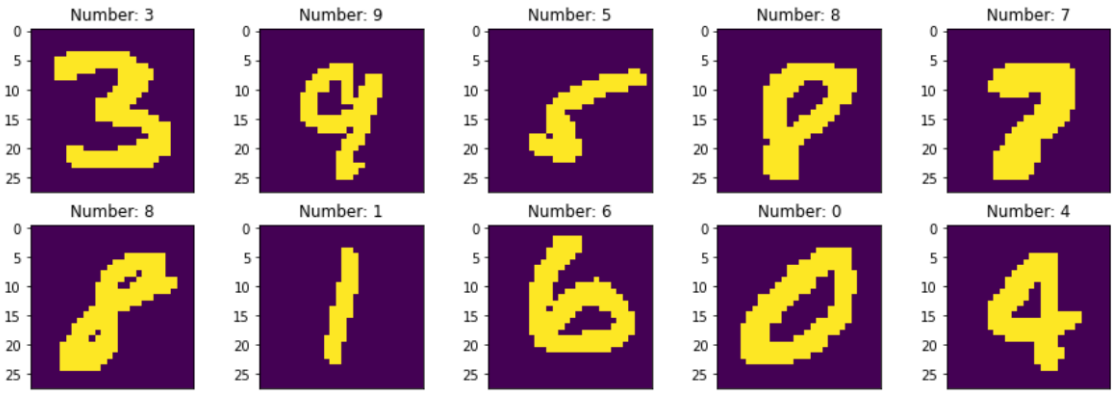


图2-1 Sample visualization

Step 5 Define a network.

We define a simple fully-connected network to implement image recognition. The network has only three layers:

The first layer is a fully-connected layer in the shape of 784 x 512.

The second layer is a fully-connected layer in the shape of 512 x 128.

The last layer is an output layer in the shape of 128 x 10.

To use MindSpore for neural network definition, inherit **mindspore.nn.cell.Cell**. **Cell** is the base class of all neural networks (such as **Conv2d**).

Define each layer of a neural network in the **__init__** method in advance, and then define the **construct** method to complete the forward construction of the neural network. The network layers are defined as follows:

Code:

```
# Create a deep neural network (DNN) model. The model consists of three fully-connected layers. The final output layer uses softmax for classification (10 classes represented by numbers from 0 to 9)
class ForwardNN(nn.Cell):
    def __init__(self):
        super(ForwardNN, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Dense(784, 512, activation='relu')
        self.fc2 = nn.Dense(512, 128, activation='relu')
        self.fc3 = nn.Dense(128, 10, activation=None)


    def construct(self, input_x):
        output = self.flatten(input_x)
        output = self.fc1(output)
        output = self.fc2(output)
        output = self.fc3(output)
        return output
```

Step 6 Define a loss function and an optimizer.

A loss function is also called an objective function and is used to measure the difference between a predicted value and an actual value. Deep learning reduces the loss value by continuous iteration. Defining a good loss function can effectively improve the model performance.

An optimizer is used to minimize the loss function, improving the model during training.

After the loss function is defined, the weight-related gradient of the loss function can be obtained. The gradient is used to indicate the weight optimization direction for the optimizer, improving model performance. Loss functions supported by MindSpore include **SoftmaxCrossEntropyWithLogits**, **L1Loss**, and **MSELoss**.

SoftmaxCrossEntropyWithLogits is used in this example.

MindSpore provides the callback mechanism to execute custom logic during training. The following uses **ModelCheckpoint** provided by the framework as an example. **ModelCheckpoint** can save the network model and parameters for subsequent fine-tuning.

Code:

```
# Create a network, a loss function, validation metric, and optimizer, and set related hyperparameters.
lr = 0.001
num_epoch = 10
momentum = 0.9

net = ForwardNN()
loss = nn.loss.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
metrics={"Accuracy": Accuracy()}
opt = nn.Adam(net.trainable_params(), lr)
```

Step 7 Start training.

The training process refers to a process in which training dataset is transferred to a network for training and optimizing network parameters. In the MindSpore framework, the **.train** method is used to complete this process.

Code:

```
# Build a model.
model = Model(net, loss, opt, metrics)
config_ck = CheckpointConfig(save_checkpoint_steps=1875, keep_checkpoint_max=10)
ckpoint_cb = ModelCheckpoint(prefix="checkpoint_net",directory = "./ckpt" ,config=config_ck)
# Generate a dataset.
ds_eval = create_dataset(False, batch_size=32)
ds_train = create_dataset(batch_size=32)
# Train the model.
loss_cb = LossMonitor(per_print_times=1875)
time_cb = TimeMonitor(data_size=ds_train.get_dataset_size())
print("============== Starting Training ==============")
model.train(num_epoch, ds_train,callbacks=[ckpoint_cb,loss_cb,time_cb ],dataset_sink_mode=False)
```

Loss values are displayed during training, as shown in the following. Although loss values may fluctuate, they gradually decrease and the accuracy gradually increases in general. Loss values displayed each time may be different because of their randomicity. The following is an example of loss values output during training:

```
============== Starting Training ==============
epoch: 1 step: 1875, loss is 0.06333521
epoch time: 18669.680 ms, per step time: 9.957 ms
epoch: 2 step: 1875, loss is 0.07061358
epoch time: 21463.662 ms, per step time: 11.447 ms
epoch: 3 step: 1875, loss is 0.043515638
epoch time: 25836.919 ms, per step time: 13.780 ms
epoch: 4 step: 1875, loss is 0.03468642
epoch time: 25553.150 ms, per step time: 13.628 ms
epoch: 5 step: 1875, loss is 0.03934026
epoch time: 27364.246 ms, per step time: 14.594 ms
epoch: 6 step: 1875, loss is 0.0023852987
epoch time: 31432.281 ms, per step time: 16.764 ms
epoch: 7 step: 1875, loss is 0.010915326
epoch time: 33697.183 ms, per step time: 17.972 ms
epoch: 8 step: 1875, loss is 0.011417691
epoch time: 29594.438 ms, per step time: 15.784 ms
epoch: 9 step: 1875, loss is 0.00044568744
epoch time: 28676.948 ms, per step time: 15.294 ms
epoch: 10 step: 1875, loss is 0.071476705
epoch time: 34999.863 ms, per step time: 18.667 ms
```

Step 8 Validate the model.

In this step, the original test set is used to validate the model.

Code:

```
# Use the test set to validate the model and print the overall accuracy.
metrics=model.eval(ds_eval)
print(metrics)
```

Output:

```
{'Accuracy': 0.9740584935897436}
```

Huawei AI Certification Training

HCIA-AI

ModelArts Experiment Guide

ISSUE:3.0

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

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

Notice

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

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

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 is an integral part of the company's "Platform + Ecosystem" strategy, it supports the ICT infrastructure featuring "Cloud-Pipe-Device". It evolves to reflect the latest trends of ICT development. Huawei Certification consists of two categories: ICT Infrastructure, and Cloud Service & Platform.

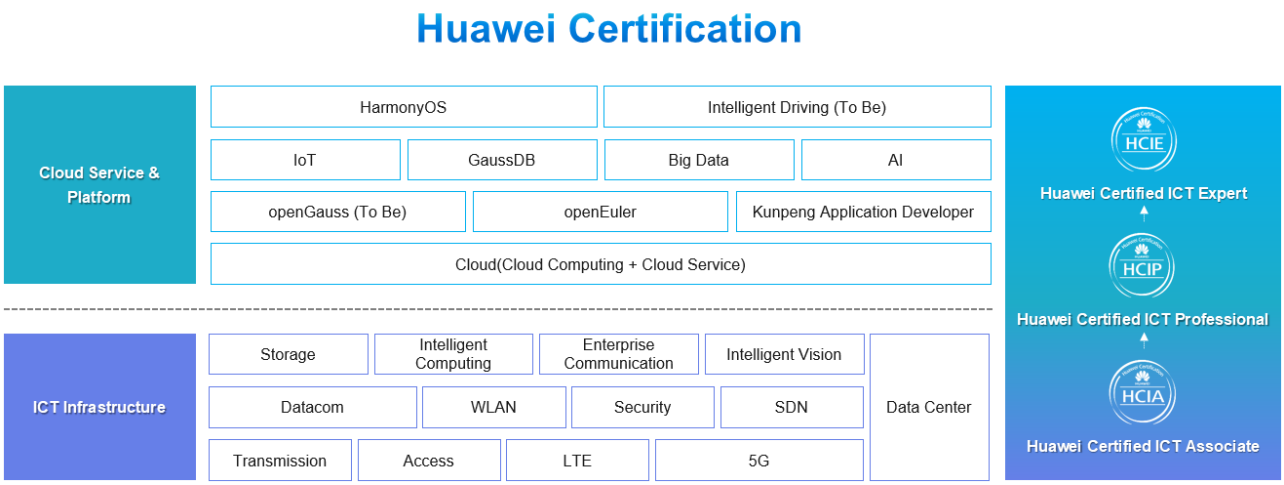
Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

With its leading talent development system and certification standards, Huawei is committed to developing ICT professionals in the digital era, building a healthy ICT talent ecosystem.

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



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.

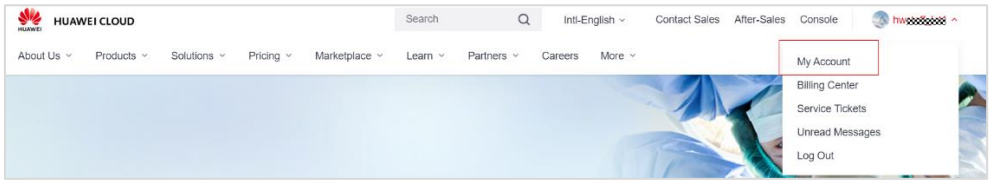


Figure 1-1 Huawei Cloud Homepage

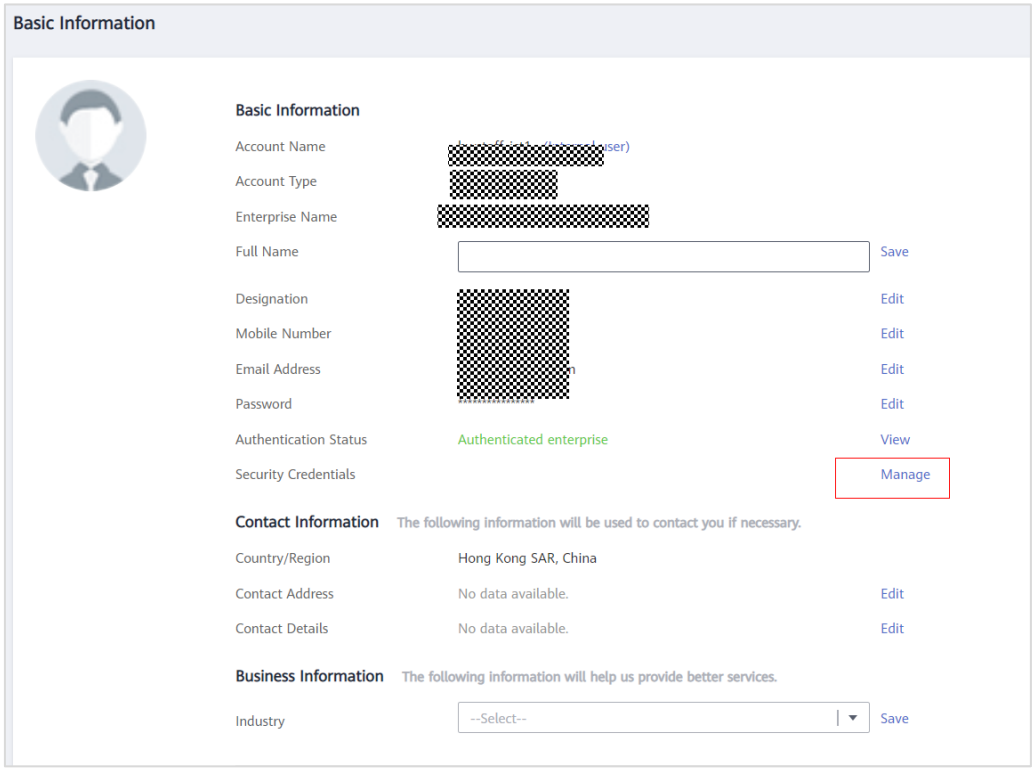


Figure 1-2 Basic Information Page

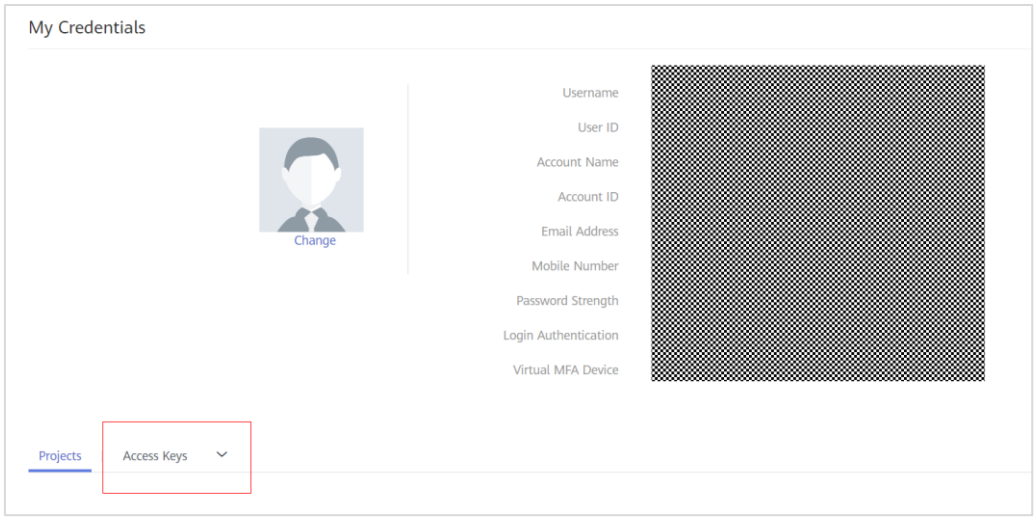


Figure 1-3 My Credentials Page

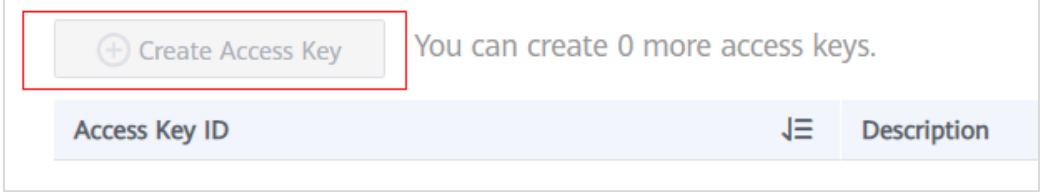


Figure 1-4 Create Access Key

- 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.

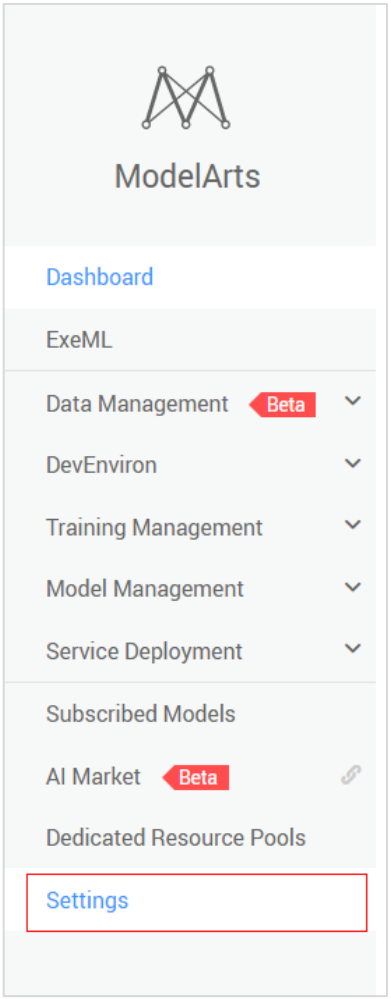


Figure 1-5 ModelArts Settings Page

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.

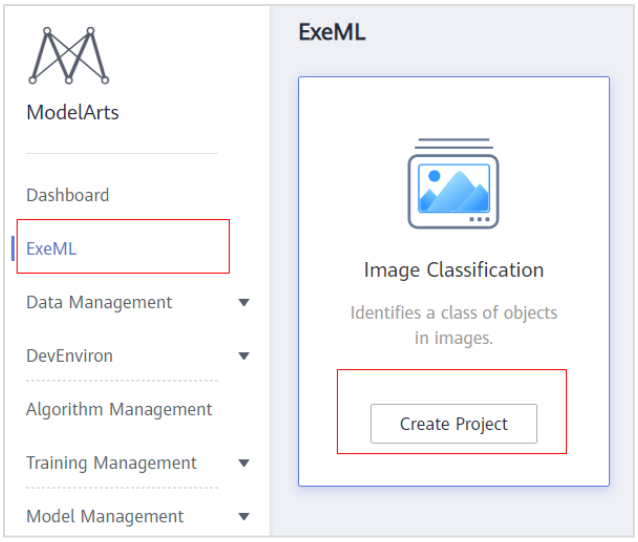


Figure 1-6 ModelArts ExeML

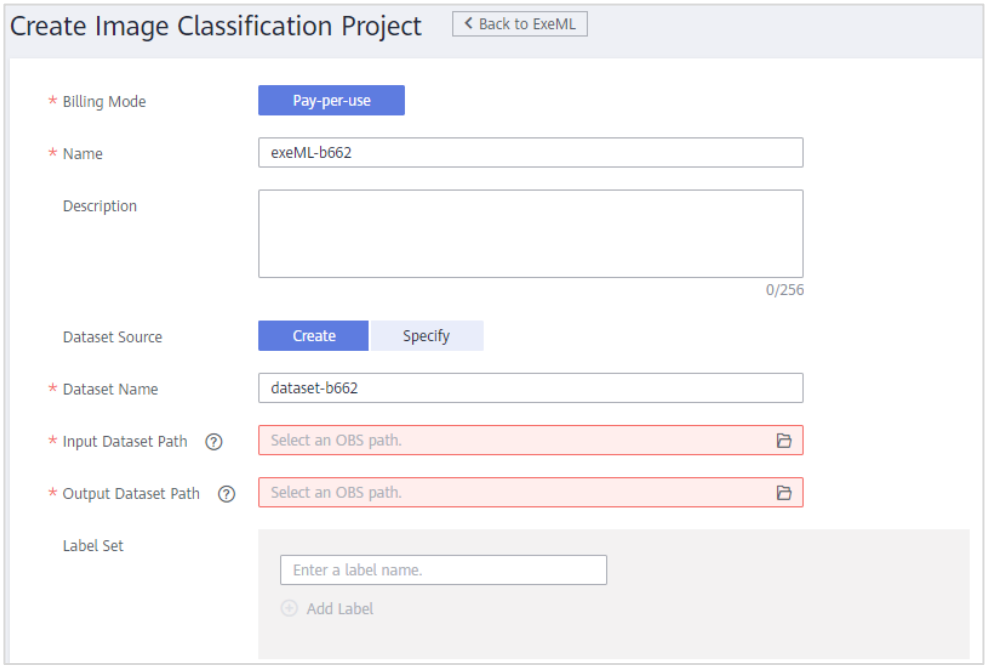


Figure 1-7 Create Image Classification Project

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.

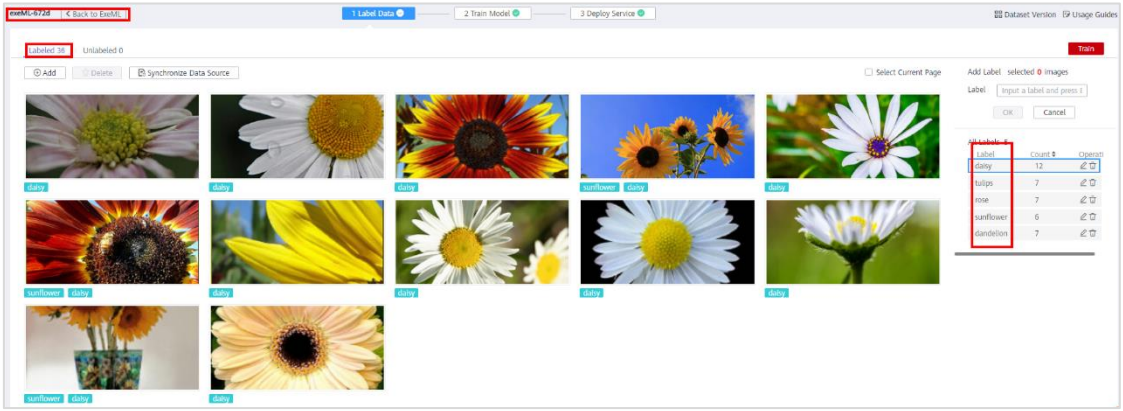


Figure 1-8 Upload Images

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.

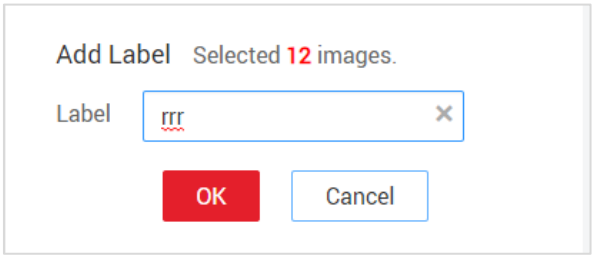


Figure 1-9 Label the Images

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.

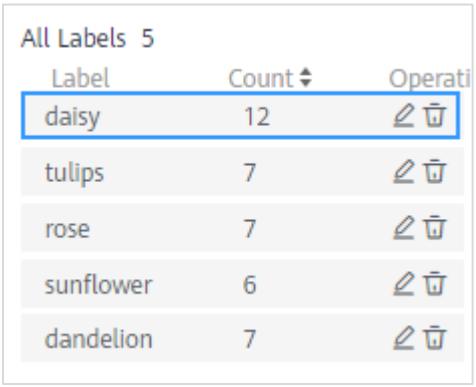


Figure 1-10 Delete or Modify Label in One Image

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.**

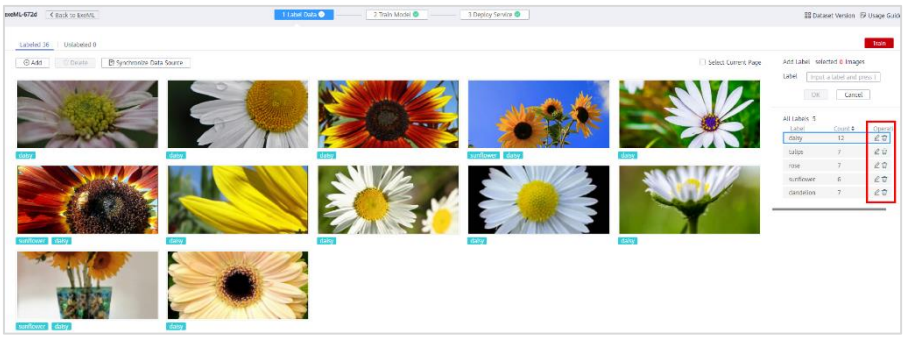


Figure 1-11 Delete or Modify a Label in Multiple Images

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.

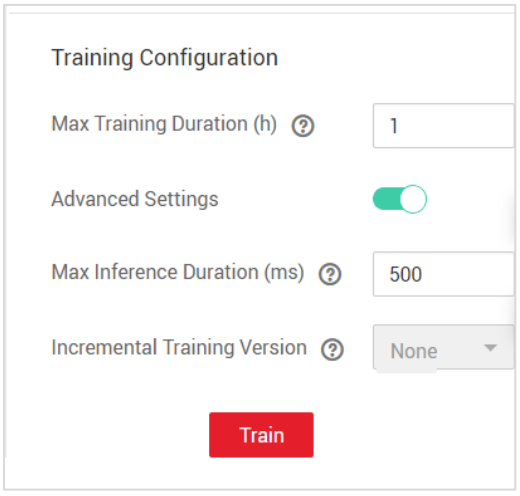


Figure 1-12 Training Configuration

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.

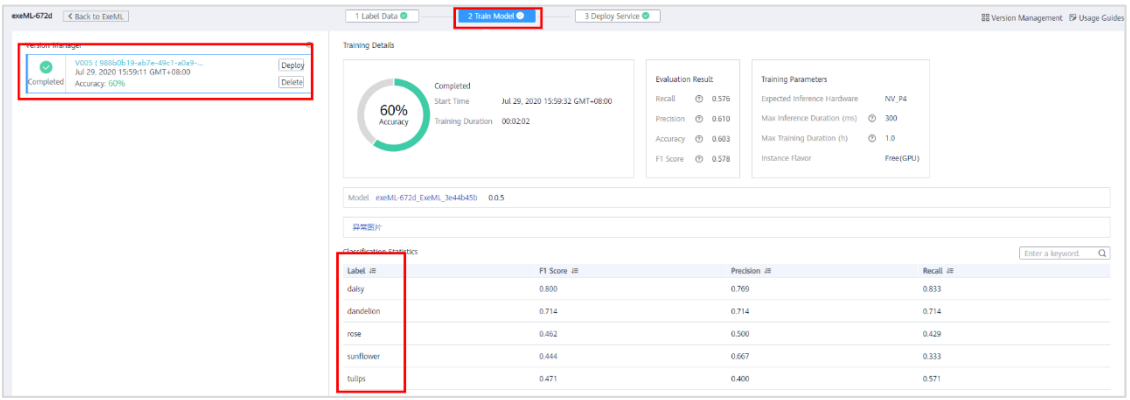


Figure 1-13 Train Model

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".

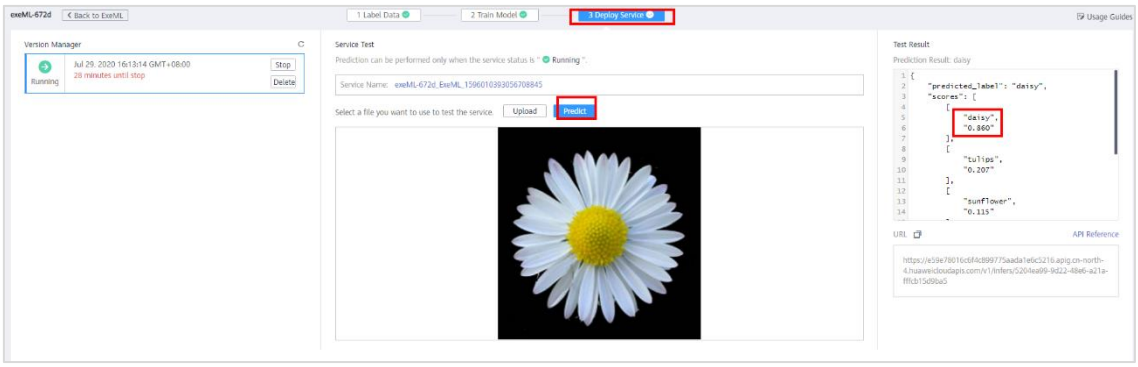


Figure 1-14 Deploy Service

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.

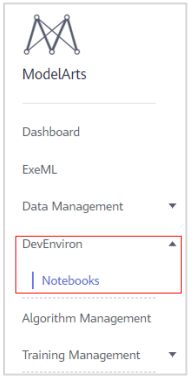


Figure 2-1 ModelArts Homepage

Click the Create button to Create the Notebook environment.

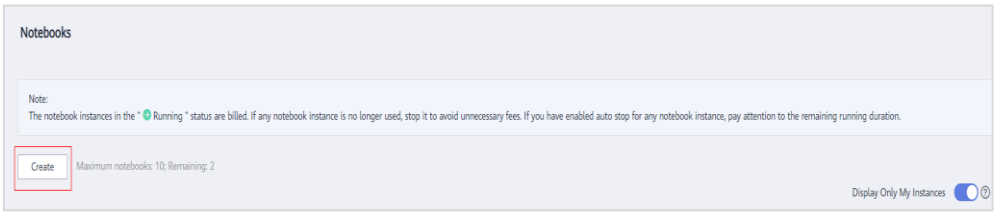


Figure 2-2 Create Notebook

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.

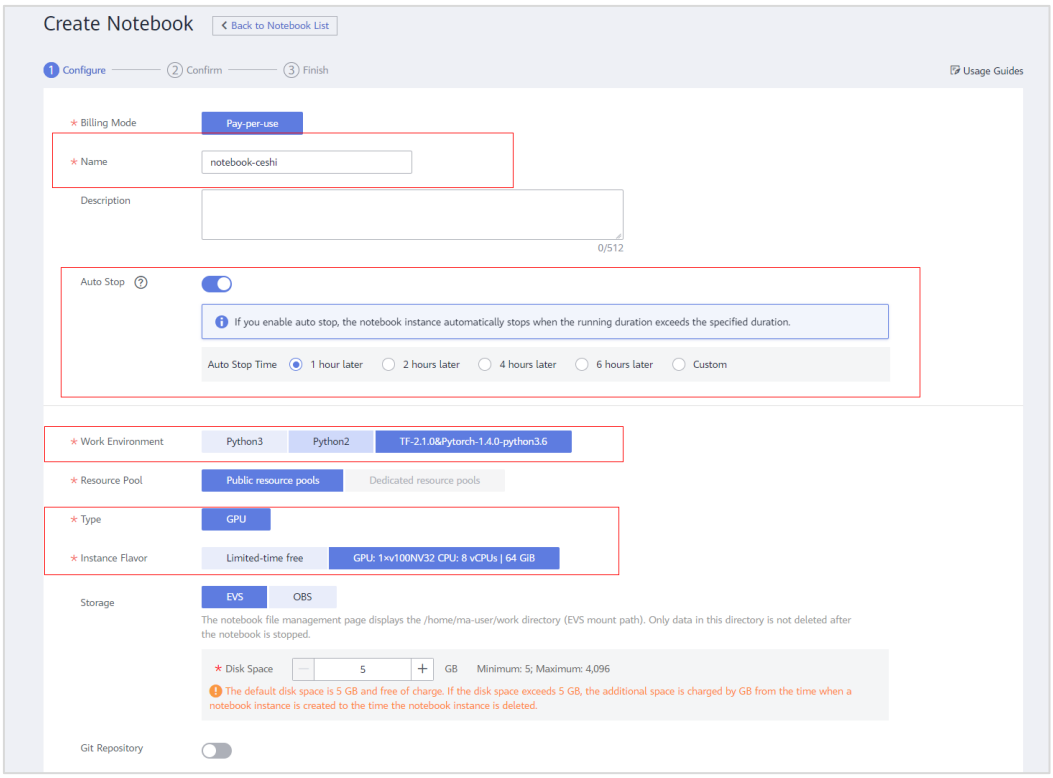


Figure 2-3 Notebook Configuration

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.

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.

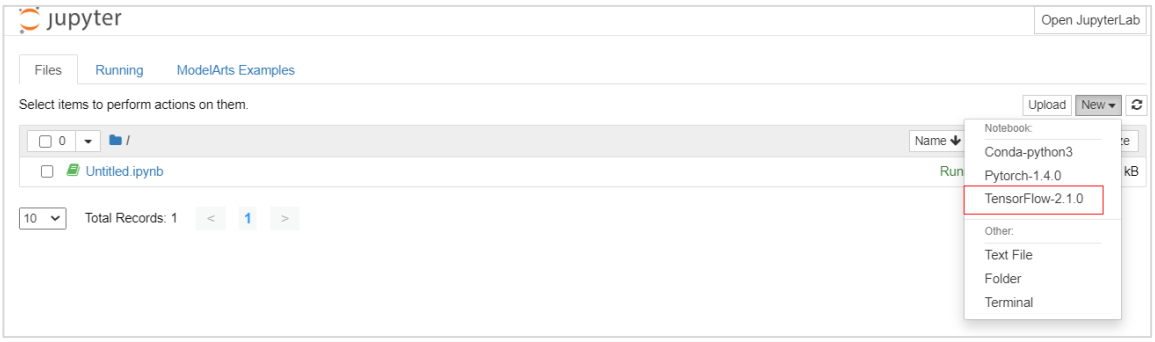


Figure 2-5 Create a New File

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

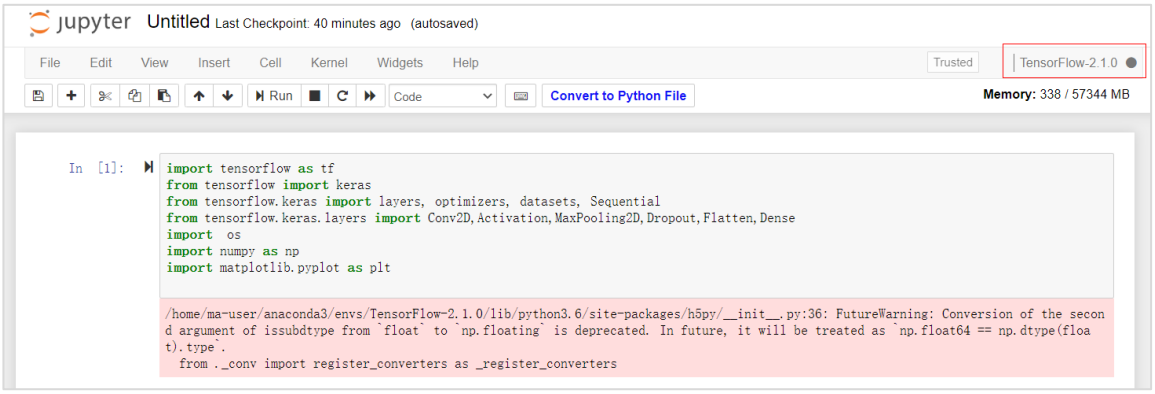


Figure 2-6 Notebook's Code Editing Interface

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:

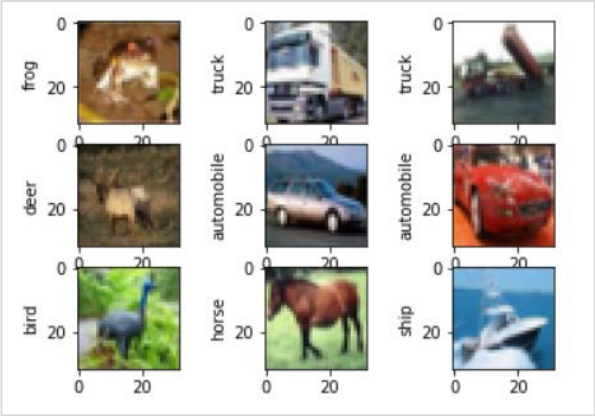


Figure 2-7 First 9 Images with Tags

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.