Huawei AI Certification Training

HCIA-AI

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

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

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

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

About This Document

Overview

This document is applicable to the candidates who are preparing for the HCIA-AI exam.

Description

This document consists of the following parts:

- Part 1: describes how to set up the Windows experiment environment.
- Part 2: describes how to set up the macOS experiment environment.
- Part 3: describes how to use the Huawei Cloud.

If you are in an area where Huawei Cloud is not accessible, the **Part 3** Using Huawei Cloud and chapter 8 in the Lab Guide can be ignored, chapter 8 is not necessary for passing the HCIA-AI V3.0 certification. Chapter 2 and 4 in the Lab Guide can be completed on the PC.

Background Knowledge Required

The readers must be able to:

- Have basic Python knowledge.
- Understand basic TensorFlow concepts.
- Command basic Python programming.

Hardware Requirements

PC Hardware Resource Requirements	
Operating System	64 bits Windows 10, MacOS Sierra or later, ubuntu18.04 or later, CentOS8.0 or later
CPU	2-core CPU, CPU frequency higher than 1 GHz
Memory	4 GB or above
GPU	Recommended but unnecessary configuration

Contents

1 Configuring the Windows Experiment Environment

1.1 Installing Anaconda

1.1.1 Introduction to Anaconda

Anaconda is a distribution of Python for scientific computing. It supports Linux, macOS, and Windows. It provides simplified package management and environment management, and easily deals with the installation issues when the system has multiple Python versions and third-party packages. Anaconda uses the conda tool/command or pip to implement package and environment management. It also comes with Python and related tools. Anaconda is a Python tool for enterprise-level big data analytics. It contains more than 720 open-source data-science packages, including data visualization, machine learning, and deep learning. It can be used for data analysis, big data and AI fields.

Anaconda provides the following features for developers:

- With Anaconda, you do not need to install Python. You only need to select the Python version when downloading Anaconda.
- With Anaconda, you only need to add a virtual environment to Anaconda when different frameworks are required to support the development. You can conduct development in different environments without worrying about compatibility issues. You can also configure environments for special projects to facilitate management.

1.1.2 Installation Procedure

Step 1 Download Anaconda.

Log in to the official Anaconda website and download the installation package **https://www.anaconda.com/distribution/#download-section**.

Select the version for Windows, macOS, or Linux. In this example, select **Windows**. Select **Python 3.7 version** (recommended) or **Python 2.7 version**, and click **64-Bit Graphical Installer** (not supported by a 32-bit computer).

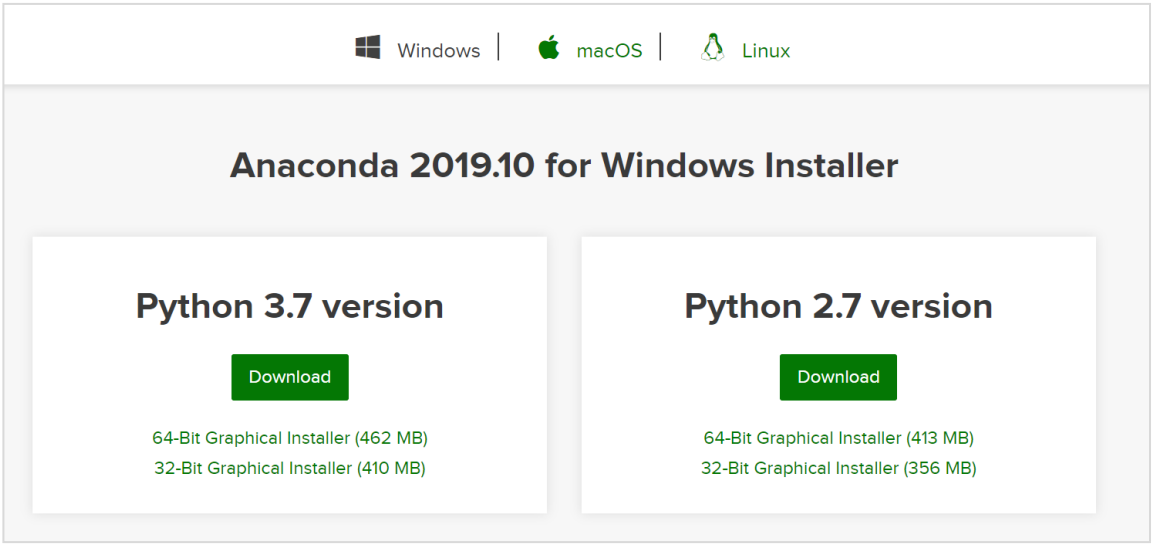


Figure 1-1

Step 2 Install Anaconda.

Double-click the downloaded **Anaconda3-x.x.x-Windows-x86_64.exe** file. In the dialog box that is displayed as shown in Figure 1-2, click **Next**.

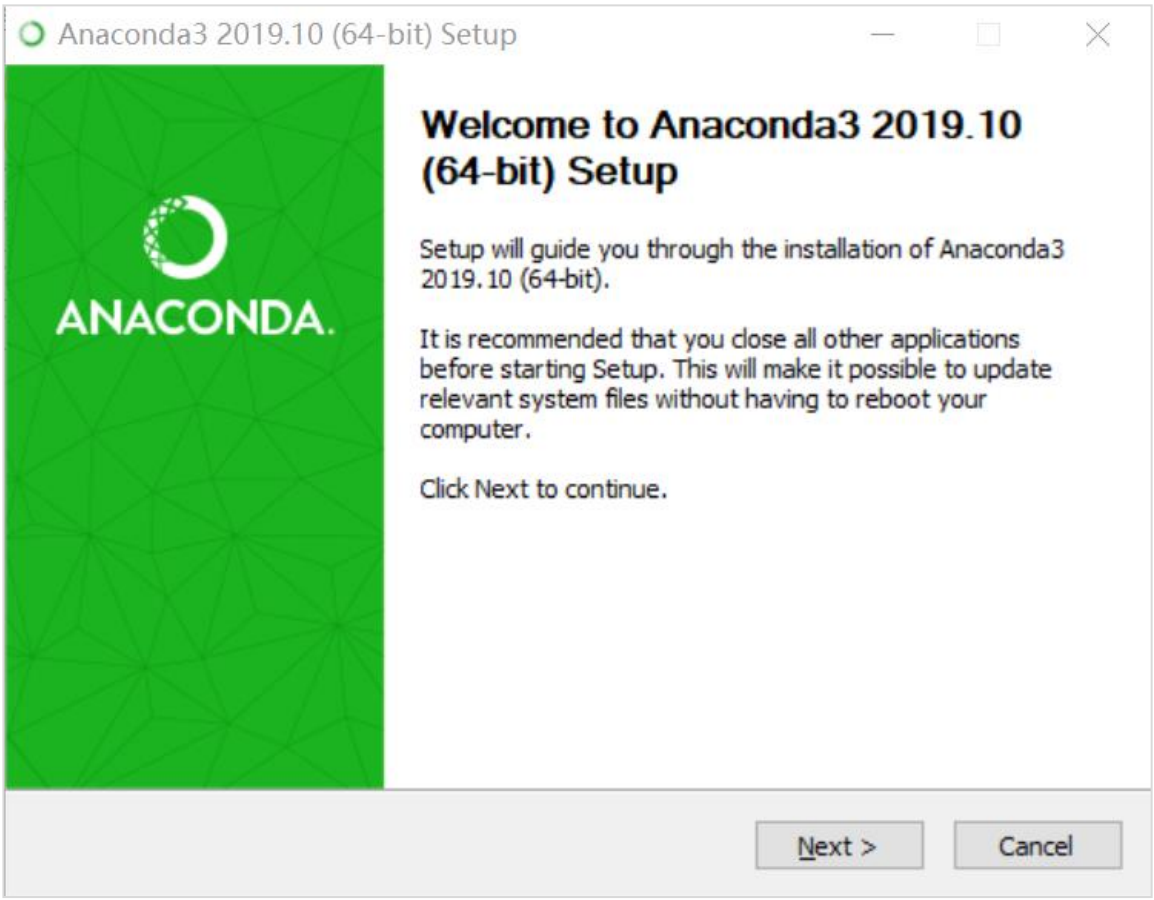


Figure 1-2

Click **I Agree**.

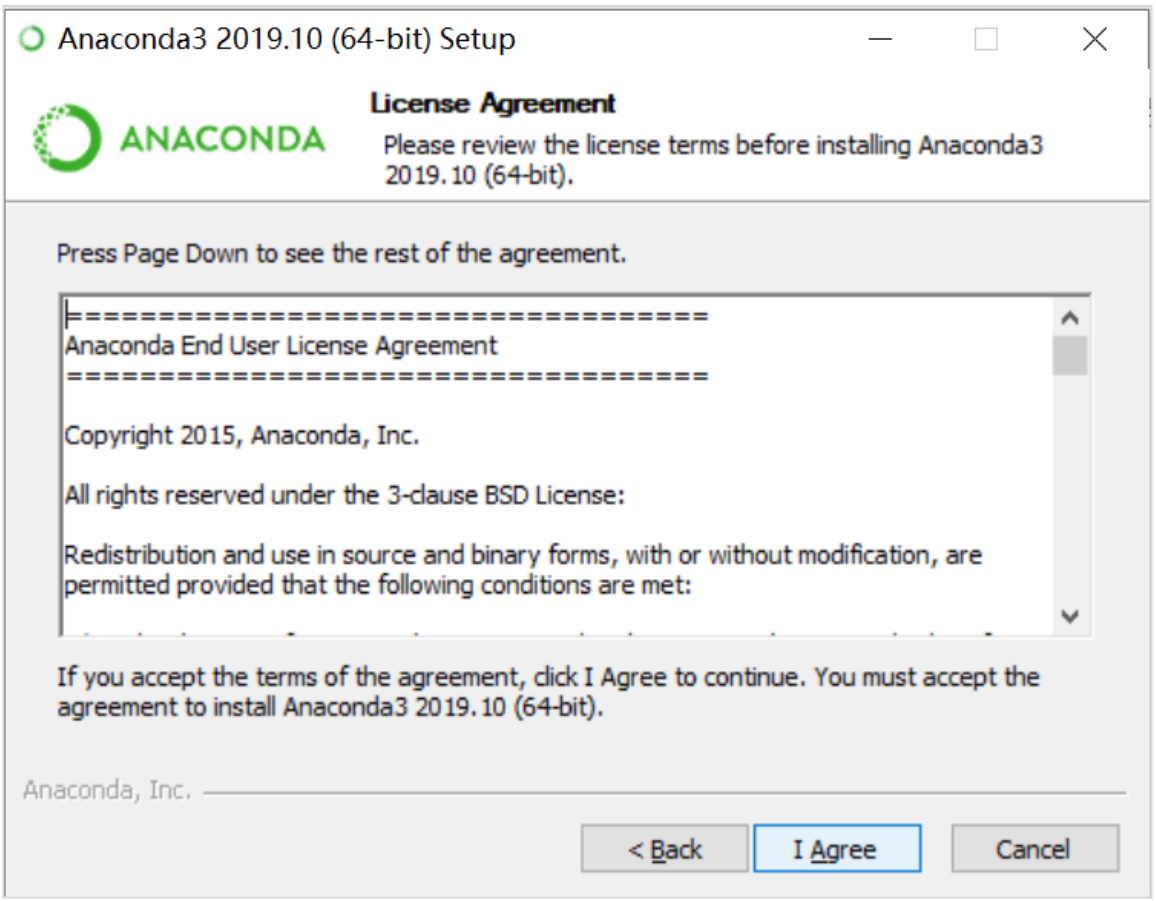


Figure 1-3

Select **Just Me** and click **Next**.

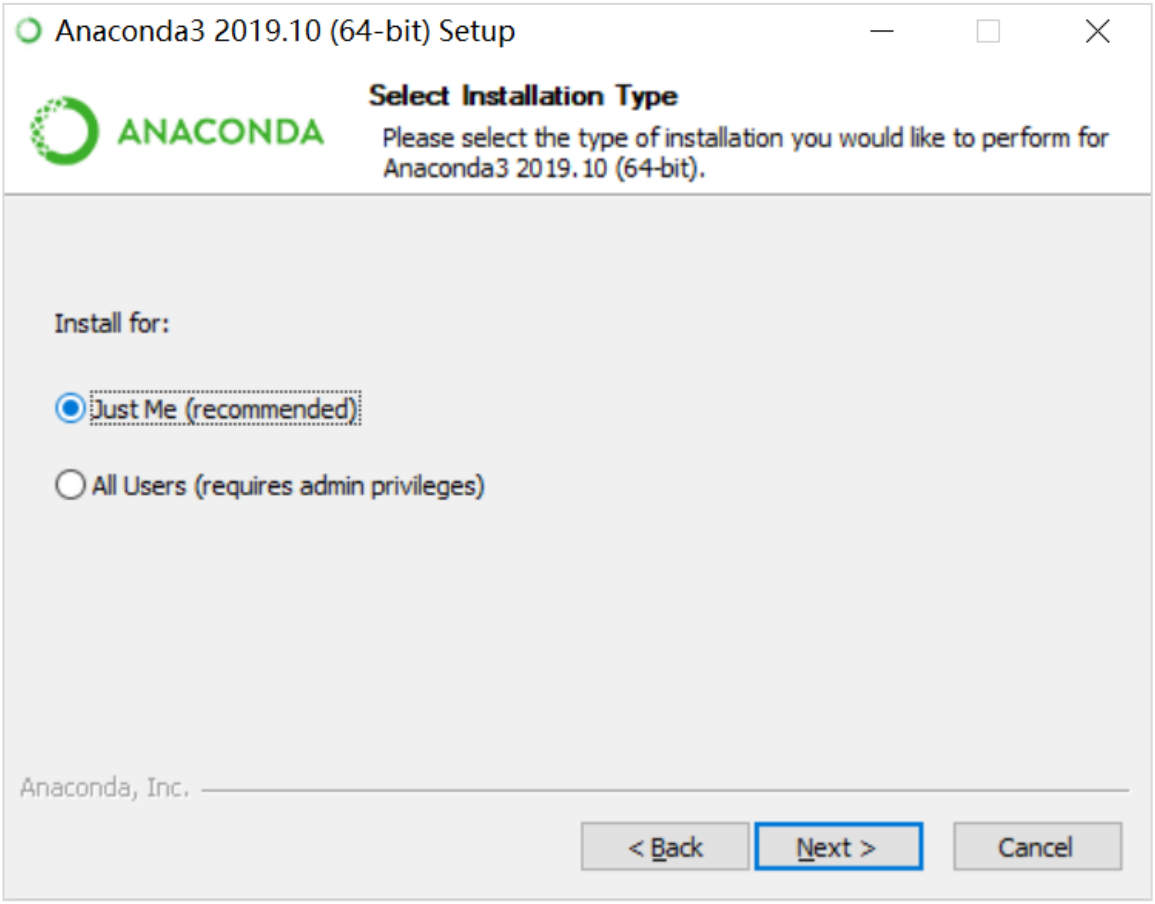


Figure 1-4

Step 3 Set the installation path.

Specify the software installation path and click **Next**.

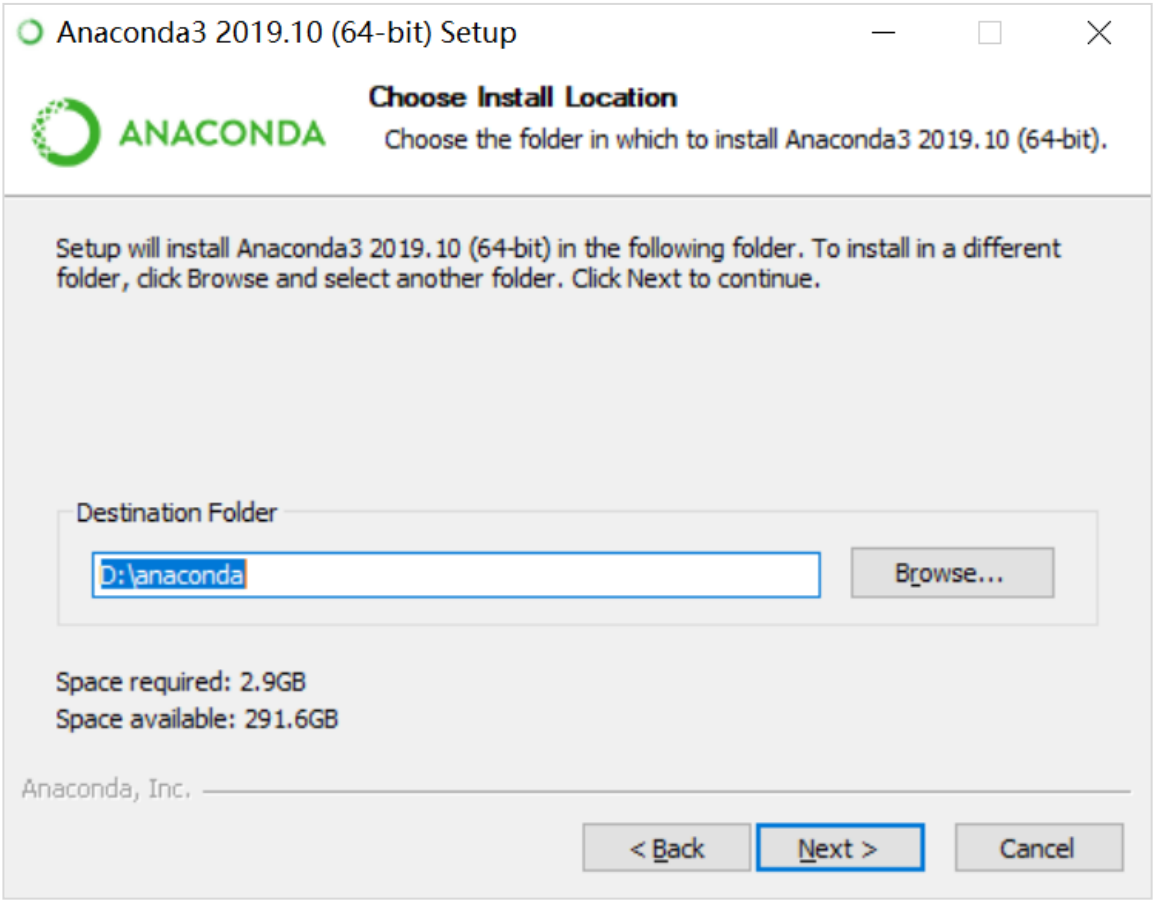


Figure 1-5

Step 4 Configure environment variables.

Select **Add Anaconda to my PATH environment variable** and **Register Anaconda as my default Python 3.7** to reduce subsequent configurations, and click **Install**. (Note: If Python of another version has been installed on the computer, you are advised to remove it and install Anaconda. If it is not removed, do not select the two options. Otherwise, a path error may occur.)

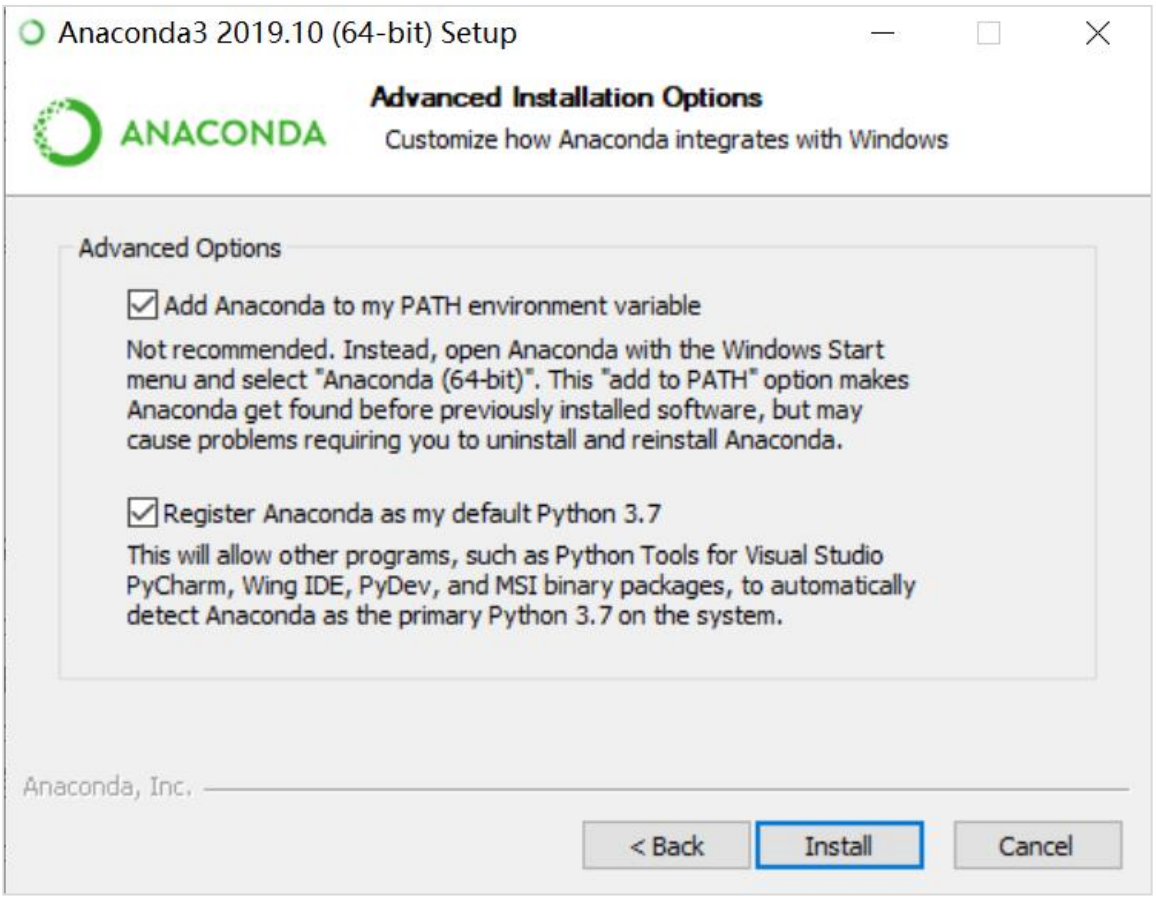


Figure 1-6

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

1.2 Changing the Source

1.2.1 Changing the **conda** Command Source

Generally, the **conda** or **pip** command is used to download the required module package from a server outside China. This consumes a lot of time when the network speed is low. Therefore, you are advised to download packages from a source in China.

On the CLI, run the following commands to change the **conda** command source to 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/
```

Example:

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

Figure 1-7

Note: Use **conda** to install Jupyter Notebook or Spyder. After the installation, the corresponding icon is displayed in the start menu, and the text in the brackets next to the name indicates the virtual environment to which the IDE belongs. The following shows an example.

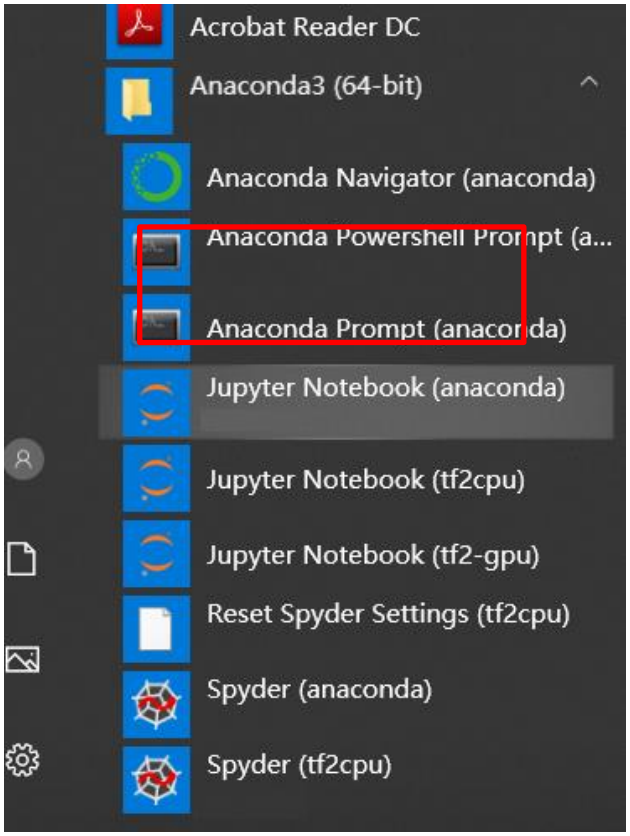


Figure 1-8

1.2.2 Changing the **pip** Command Source

You can change the source of the **pip** command temporarily (for the current command only) or change the default source (after the change, all commands download resources from the new source by default).

1.2.2.1 Changing the pip Source Temporarily

Add **-i *source website*** to the end of the command, for example:

```
pip install tensorflow -i http://pypi.douban.com/simple
```

If an error is reported, add **--trusted-host *source website*** to the end of the command, for example:

```
pip install tensorflow -i http://pypi.douban.com/simple --trusted-host pypi.douban.com
```

1.2.2.2 Changing the Default pip Source

Create a **pip** folder in the **C:\Users*User name*** directory, create a .txt file in the **pip** folder, and add the following to the file:

```
[global]
index-url = http://pypi.douban.com/simple/
[install]
trusted-host =pypi.douban.com
```

Then, save the file as **pip.ini** and select **All Files** for **Save as type**.

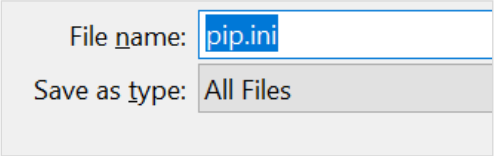


Figure 1-9

1.3 Installing TensorFlow

1.3.1 Introduction to TensorFlow

TensorFlow is Google's second-generation artificial intelligence learning system based on DistBelief. Its name derives from its operating principle. A tensor indicates an N-dimensional array, and a flow indicates calculation based on the dataflow graph. A tensor flow is a calculation process of a tensor from one end of the graph to the other end. TensorFlow is a system that transmits complex data structures to the AI neural network for analysis and processing.

The latest version TensorFlow 2.1.0 focuses on simplicity and ease of use and provides the following new features:

- Keras module added
- Eager execution of the dynamic graph mechanism
- Support for more platforms and languages
- Removal of discarded APIs and reduction of duplicate APIs
- Compatibility and continuity

TensorFlow 2.0 and 2.1.0 are dominant versions used currently. TensorFlow 2.0 is stable, and TensorFlow 2.1.0 is an optimized version released in November 2019. There are minor differences between the two versions. You can select a version as required. The following describes how to install the 2.0 and 2.1.0 releases for CPU and GPU.

1.3.2 Installing TensorFlow 2.1.0 for CPU

Step 1 Install TensorFlow 2.1.0 for CPU.

Open the Anaconda Terminal window from the start menu.

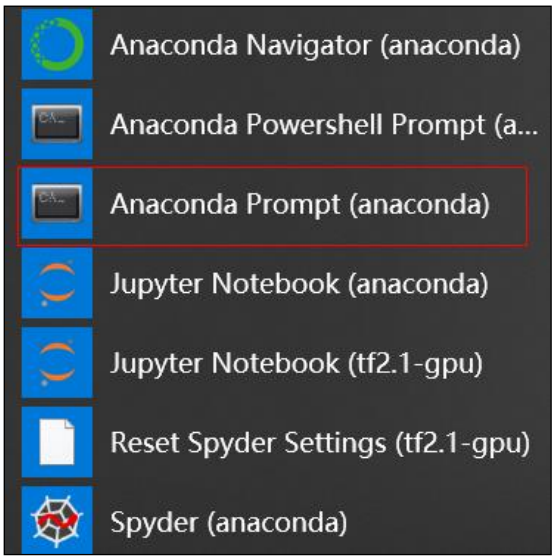


Figure 1-10

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

Step 2 Run the pip install **tensorflow-cpu** command.

```
(tf2.1) C:\Users\WWX697589>pip install tensorflow-cpu
Looking in indexes: http://pypi.douban.com/simple/
Collecting tensorflow-cpu
  Downloading http://pypi.doubanio.com/packages/57/87/1f69a143e02dc0df65228ed8e0e624c47f8ad6474339ea0b005248bb2457/tensorflow_cpu-2.1.0-cp37-cp37m-win_amd64.whl (96.8 MB)
     |████████████████████████████████| 56.3 MB 3.3 MB/s eta 0:00:13
```

Figure 1-11

The installation is successful if the following information is displayed:

```
Successfully installed absl-py-0.9.0 astor-0.8.1 cachetools-4.0.0 chardet-3.0.4 gast-0.2.2 google-auth-1.11.2 google-auth-oauthlib-0.4.1 google-pasta-0.1.8 grpcio-1.27.2 h5py-2.10.0 idna-2.9 keras-applications-1.0.8 keras-preprocessing-1.1.0 markdown-3.2.1 numpy-1.18.1 oauthlib-3.1.0 opt-einsum-3.2.0 protobuf-3.11.3 pyasn1-0.4.8 pyasn1-modules-0.2.8 requests-2.23.0 requests-oauthlib-1.3.0 rsa-4.0 scipy-1.4.1 six-1.14.0 tensorboard-2.1.1 tensorflow-2.1.0 tensorflow-estimator-2.1.0 termcolor-1.1.0 urllib3-1.25.8 werkzeug-1.0.0 wrapt-1.12.0
```

Figure 1-12

Anaconda installed comes with a Python module required in this experiment. You do not need to install pandas, numpy, or scikit-image.

Step 3 If you need to install a module, run the **pip install** *software package* command. For example, run **pip install matplotlib**.

```
(tf2) C:\Users\WWX697589>pip install matplotlib
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple/
Collecting matplotlib
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/93/22/a3cef48d25ad94f540bb3dd91490fb22afe0911acc3390b1929527ae4f/matplotlib-3.1.3-cp37-cp37m-win_amd64.whl (9.1 MB)
     |████████████████████████████████| 9.1 MB 3.3 MB/s
Collecting kiwisolver>=1.0.1
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/c6/ea/e5474014a13ab2dcb5056608e0716c600c3d8a8bcffb10ed55ccd6aeb0/kiwisolver-1.1.0-cp37-none-win_amd64.whl (57 kB)
     |████████████████████████████████| 57 kB 4.1 MB/s
Collecting python-dateutil>=2.1
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/d4/70/d60450c3dd48ef87586924207ae8907090de0b306af2bce5d134d785cb/python_dateutil-2.8.1-py2.py3-none-any.whl (227 kB)
     |████████████████████████████████| 227 kB 6.8 MB/s
Requirement already satisfied: numpy>=1.11 in d:\anaconda\envs\tf2\lib\site-packages (from matplotlib) (1.18.1)
Collecting cycler>=0.10
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/f7/d2/e07d3ebb2bd7af696440ce7e754c59dd546ffe1bbe732c8ab68b9c8e61/cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
```

Figure 1-13

Step 4 Check the installation.

Run the **pip list** command. All Python modules installed are displayed, as shown in the following figure.

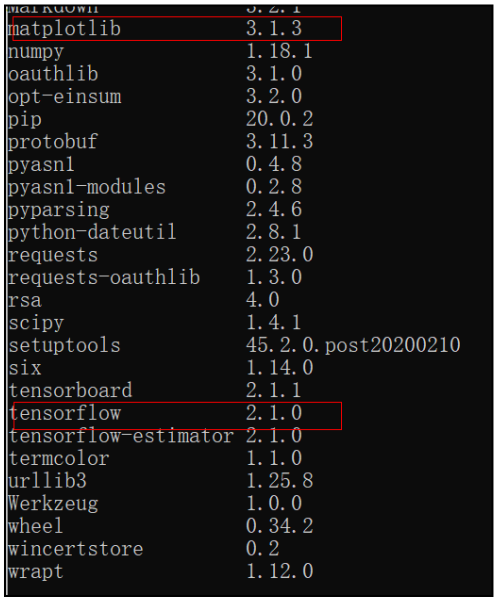


Figure 1-14

1.3.3 Installing TensorFlow 2.0.0 for CPU

Step 1 Install TensorFlow 2.0.0 for CPU.

Open the Anaconda Terminal window from the start menu.

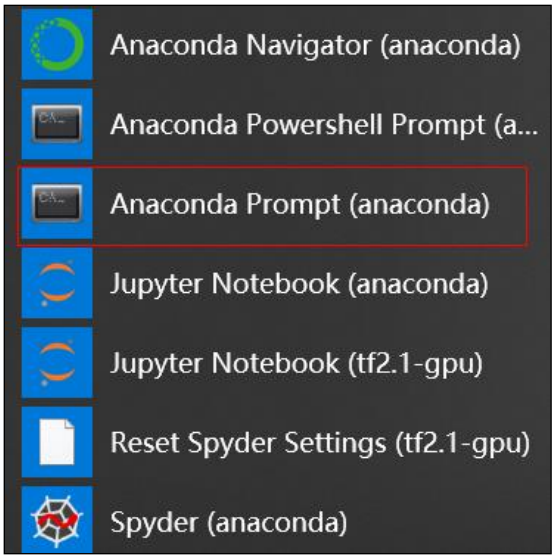


Figure 1-15

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

Step 2 Run the **pip install tensorflow==2.0.0 command**.

```
(tf2) C:\Users\WWX697589>pip install tensorflow==2.0.0
Looking in indexes: http://pypi.douban.com/simple/
Collecting tensorflow==2.0.0
  Downloading http://pypi.doubanio.com/packages/54/5f/e1b2d83b808f978f51b7ce109315154da3a3d4151aa59686002681f2e109/tensorflow-2.0.0-cp37-cp37m-win_amd64.whl (48.1 MB)
     |████████████████████████████████| 36.9 MB 2.2 MB/s eta 0:00:06
```

Figure 1-16

The installation is successful if the following information is displayed:

```
Successfully installed absl-py-0.9.0 astor-0.8.1 cachetools-4.0.0 chardet-3.0.4 gast-0.2.2 google-auth-1.11.2 google-auth-oauthlib-0.4.1 google-pasta-0.1.8 grpcio-1.27.2 h5py-2.10.0 idna-2.9 keras-applications-1.0.8 keras-preprocessing-1.1.0 markdown-3.2.1 numpy-1.18.1 oauthlib-3.1.0 opt-einsum-3.2.0 protobuf-3.11.3 pyasn1-0.4.8 pyasn1-modules-0.2.8 requests-2.23.0 requests-oauthlib-1.3.0 rsa-4.0 scipy-1.4.1 six-1.14.0 tensorboard-2.1.1 tensorflow-2.1.0 tensorflow-estimator-2.1.0 termcolor-1.1.0 urllib3-1.25.8 werkzeug-1.0.0 wrapt-1.12.0
```

Figure 1-17

1.3.4 Installing TensorFlow 2.1.0 for GPU

Step 1 Install TensorFlow 2.1.0 for GPU.

Open the Anaconda Terminal window from the start menu.

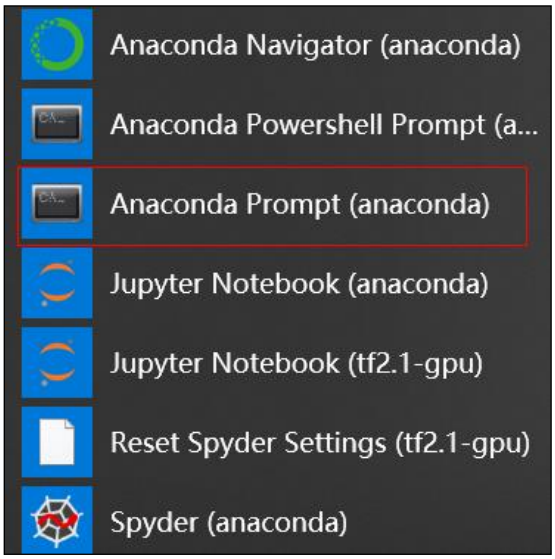


Figure 1-18

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

Step 2 Run the **pip install tensorflow** command.

```
(tf2.1-gpu) C:\Users\WWX697589>pip install tensorflow
Looking in indexes: http://pypi.douban.com/simple/
Collecting tensorflow
  Downloading http://pypi.doubanio.com/packages/34/d5/ce8c17971067c0184c9045112b755be5461d5ce5253ef65a367e1298d7c5/tensorflow-2.1.0-cp37-cp37m-win_amd64.whl (355.8 MB)
     |████████████████████████████████| 355.8 MB 6.8 MB/s
```

Figure 1-19

The installation is successful if the following information is displayed:

```
Successfully installed absl-py-0.9.0 astor-0.8.1 cachetools-4.0.0 chardet-3.0.4 gast-0.2.2 google-auth-1.11.2 google-auth-oauthlib-0.4.1 google-pasta-0.1.8 grpcio-1.27.2 h5py-2.10.0 idna-2.9 keras-applications-1.0.8 keras-preprocessing-1.1.0 markdown-3.2.1 numpy-1.18.1 oauthlib-3.1.0 opt-einsum-3.2.0 protobuf-3.11.3 pyasn1-0.4.8 pyasn1-modules-0.2.8 requests-2.23.0 requests-oauthlib-1.3.0 rsa-4.0 scipy-1.4.1 six-1.14.0 tensorboard-2.1.1 tensorflow-2.1.0 tensorflow-estimator-2.1.0 termcolor-1.1.0 urllib3-1.25.8 werkzeug-1.0.0 wrapt-1.12.0
```

Figure 1-20

Step 3 GPU acceleration can be implemented only when the local host has an independent graphics card and CUDA and cuDNN are installed. Install CUDA first. Download the required CUDA version from https://developer.nvidia.com/cuda-10.1-download-archive-update2?target_os=Windows&target_arch=x86_64&target_version=10&target_type=exenetwork.

This URL is directed to CUDA 10.1. To download other versions, click **Legacy Releases**.

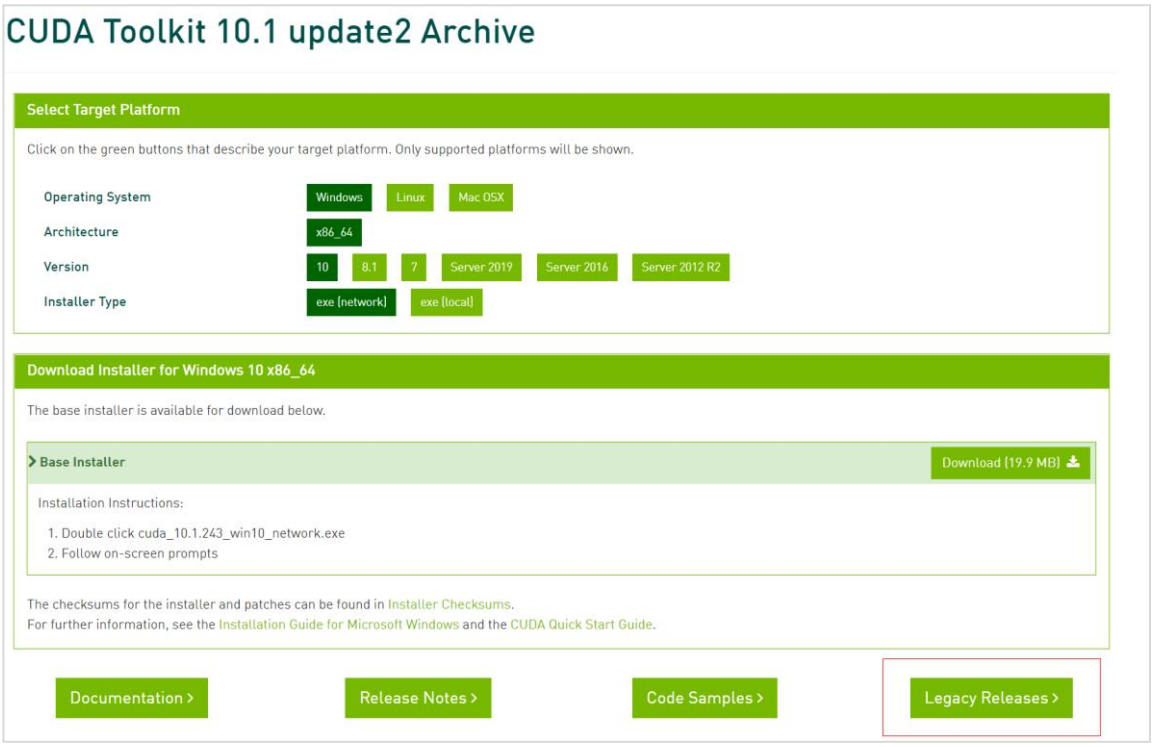


Figure 1-21

Step 4 Double-click the downloaded program to install CUDA. At the beginning, the program automatically decompresses the package to the default directory. Click **OK** to go to the next step.

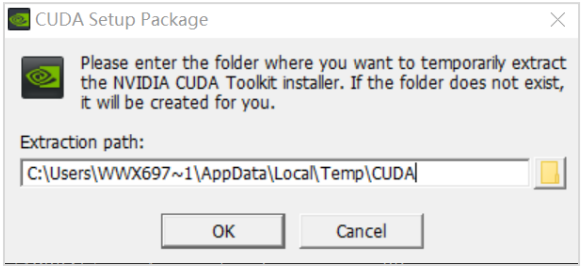


Figure 1-22

Step 5 After the decompression is complete, the installation starts. Click **AGREE AND CONTINUE**.

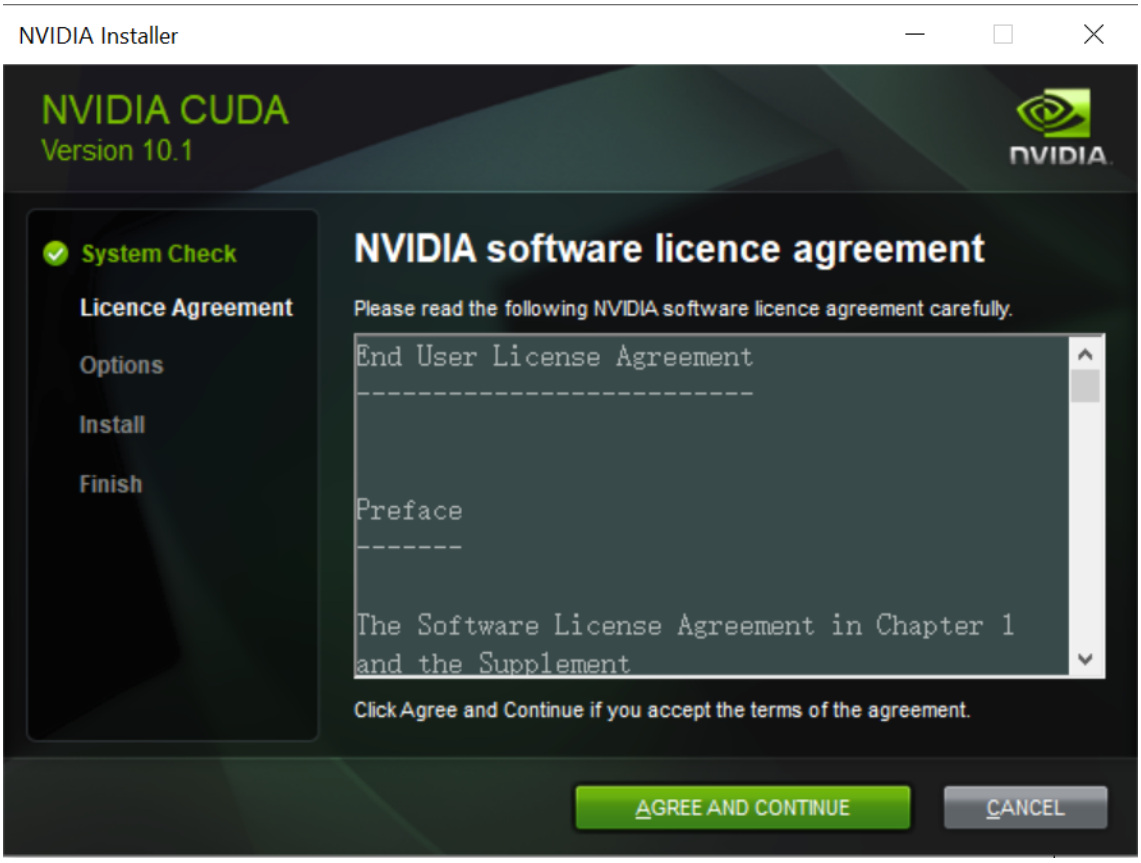


Figure 1-23

Step 6 Select **Custom** and click **NEXT**.

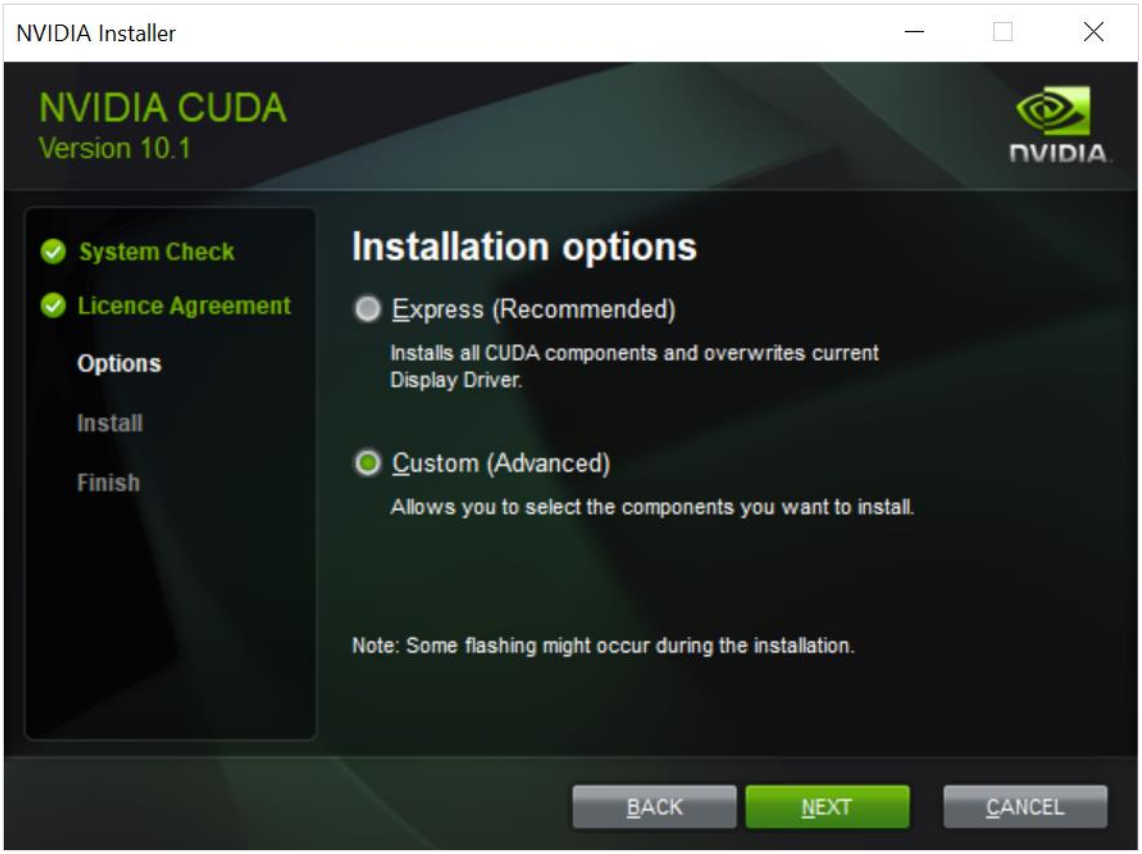


Figure 1-24

The window shown in Figure 1-25 is displayed.

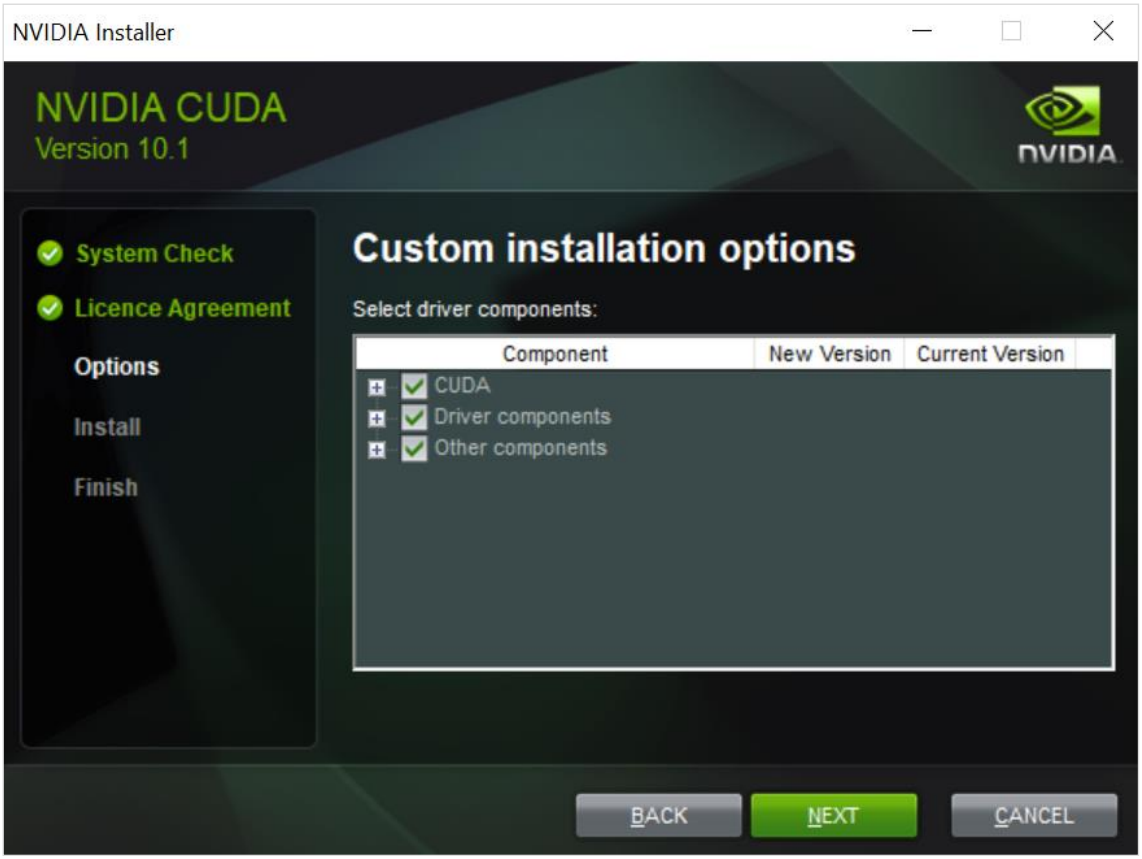


Figure 1-25

Note: If "No Visual Studio Integration" is displayed after you click **NEXT**, click the link and download Visual Studio.

Wait until the installation is complete.

Step 7 Install cuDNN. Register with the official website (https://developer.nvidia.com/cudnn) and download the cuDNN software.

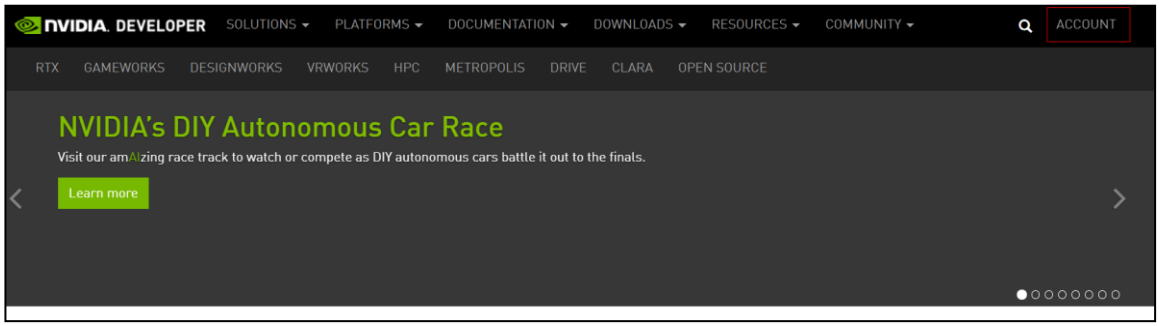


Figure 1-26

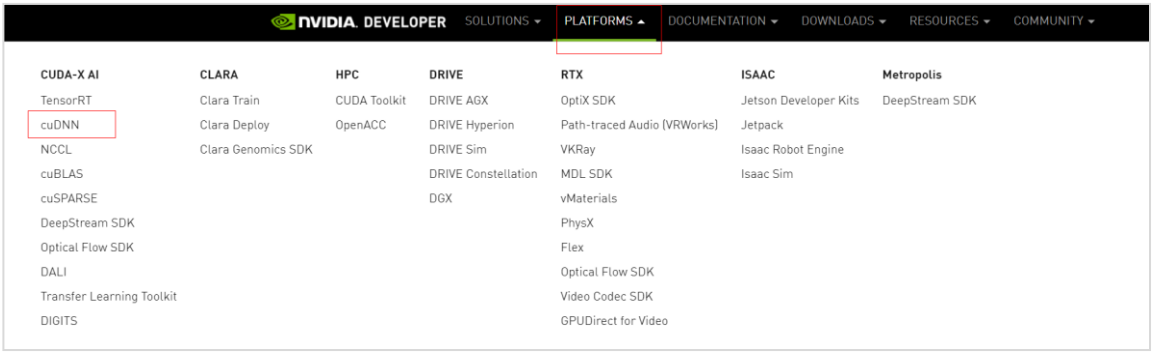


Figure 1-27

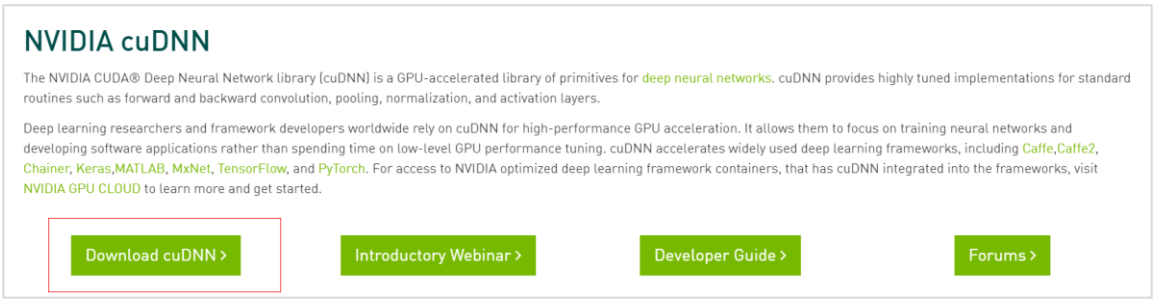


Figure 1-28

Step 8 The cuDNN version must match the CUDA version. Since CUDA 10.1 is installed, download cuDNN 7.6.5.

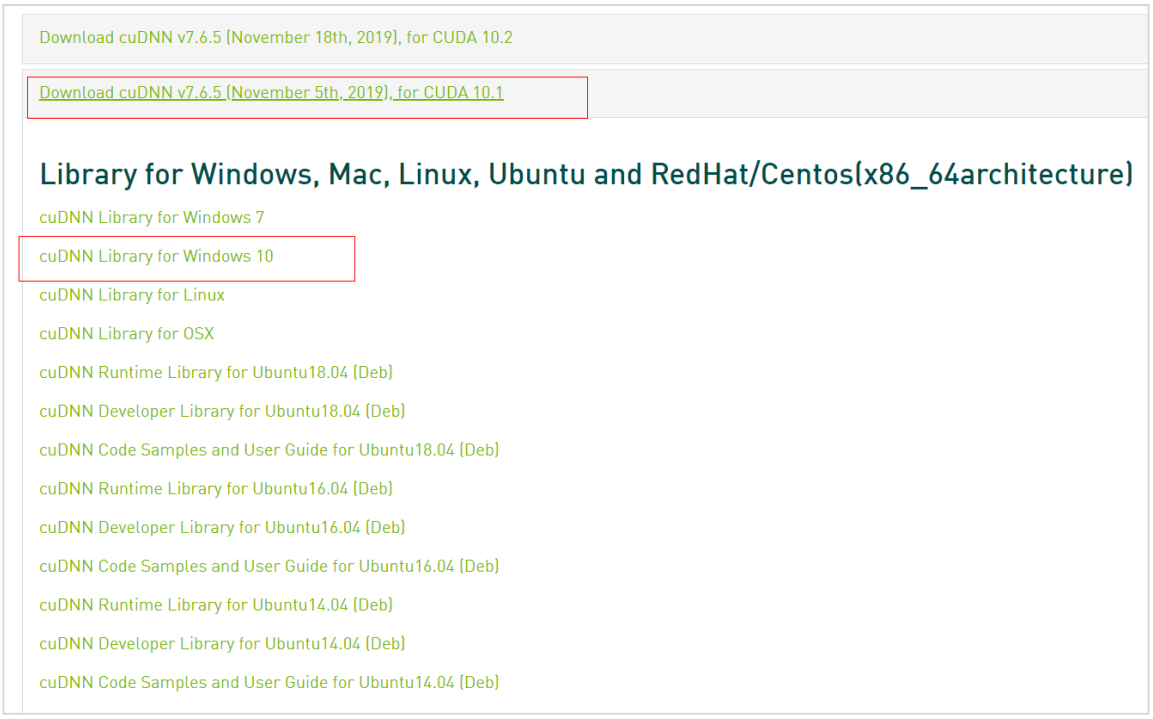


Figure 1-29

Step 9 Decompress the package downloaded.

Three folders are generated, as shown in Figure 1-30.

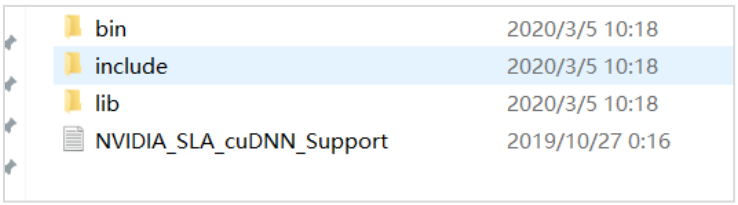


Figure 1-30

Step 10 Copy the three folders to the **CUDA** directory. If you do not change the paths, the default path is **C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1**. Replace the folders in the directory with the three folders.

Step 11 Configure the CUDA environment variables. Search for "environment variables" in the search box and click **Open**.

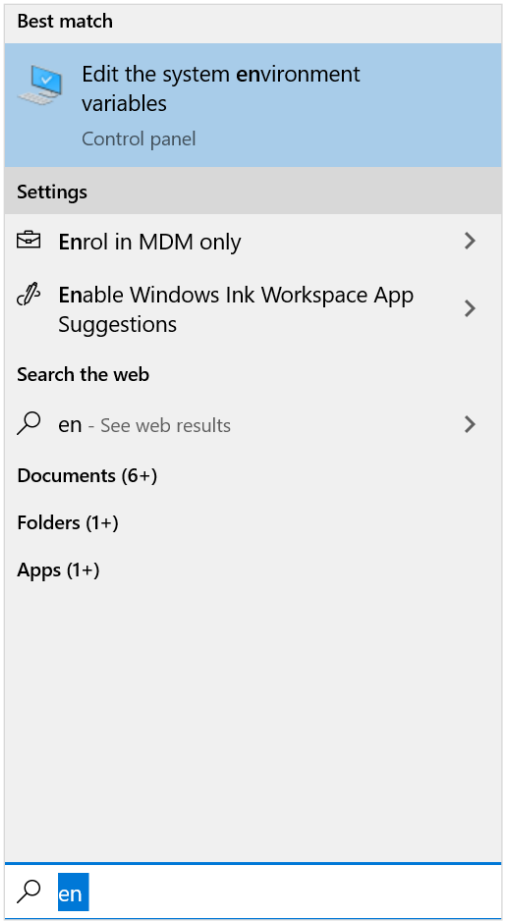


Figure 1-31

Step 12 Check whether the CUDA system variables exist, as shown in Figure 1-32.

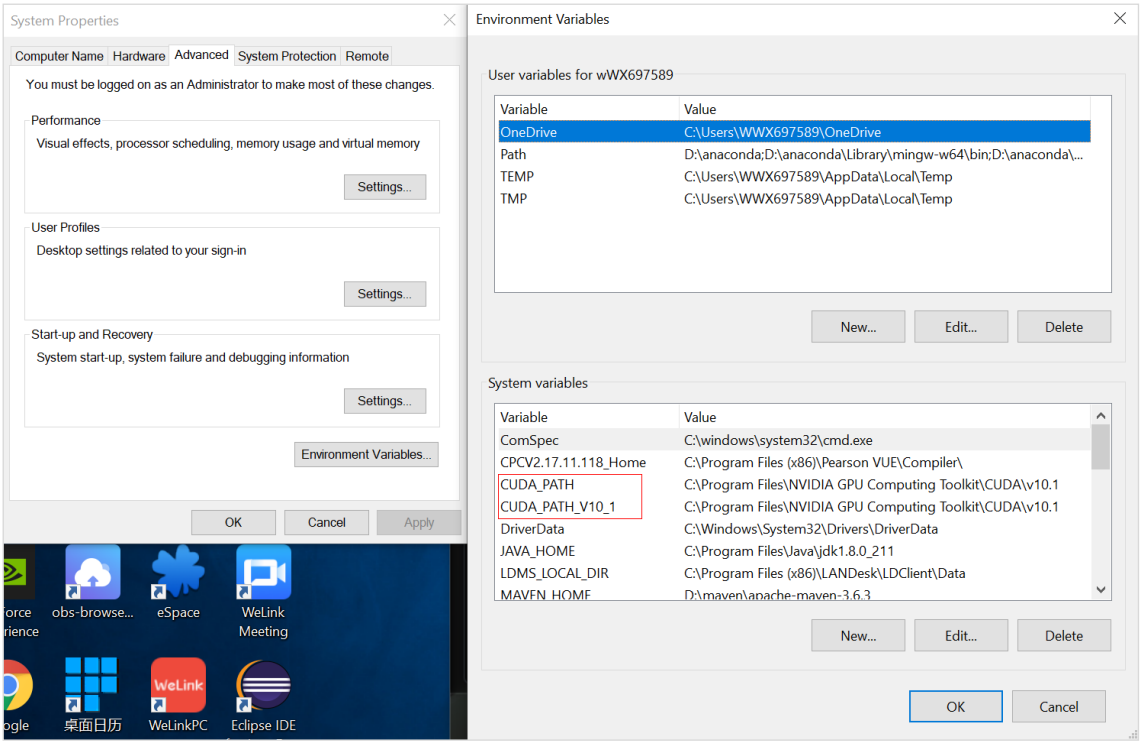


Figure 1-32

Step 13 Add the CUDA paths to **Path**. If you never change the paths, add the following paths:

- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1
- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\bin
- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\lib\x64
- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\libnvvp

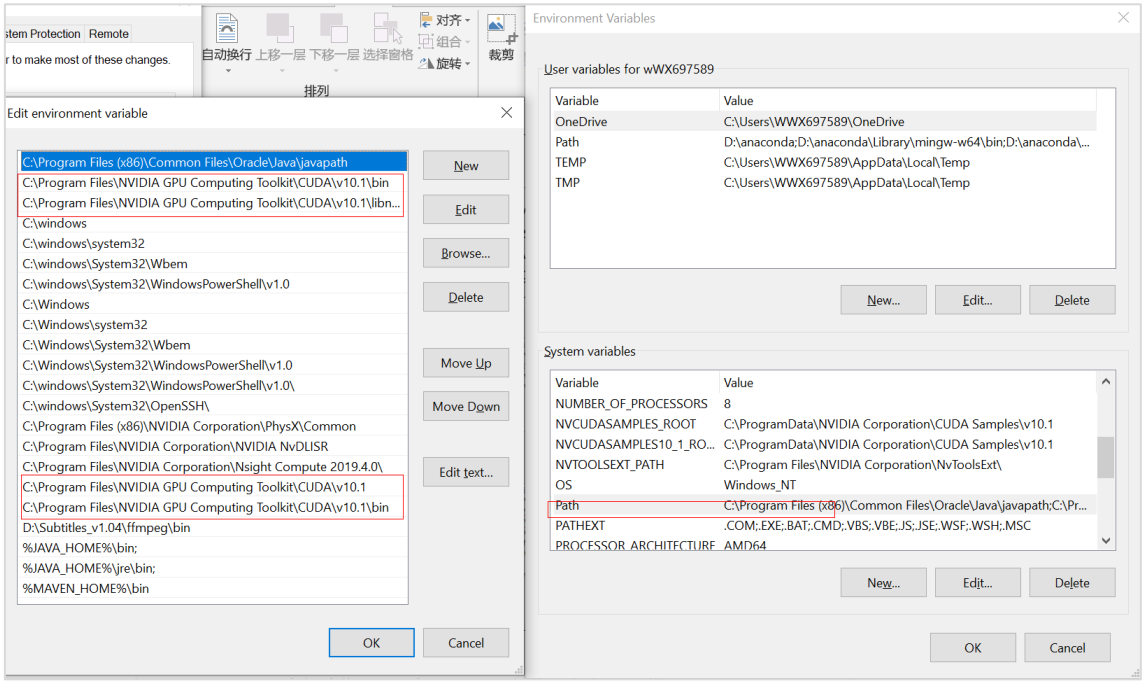


Figure 1-33

Step 14 On the CLI, run **nvcc –V**.

If information similar to Figure 1-34 is displayed, the installation is successful.

```
C:\Users\WWX697589>nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Sun_Jul_28_19:12:52_Pacific_Daylight_Time_2019
Cuda compilation tools, release 10.1, V10.1.243
```

Figure 1-34

You can also run **import tensorflow** in the GPU environment and TensorFlow for GPU version. If no error is reported, the installation is successful.

1.3.5 Installing TensorFlow 2.0.0 for GPU

Step 1 Install TensorFlow 2.0.0 for GPU.

Open the Anaconda Terminal window from the start menu.

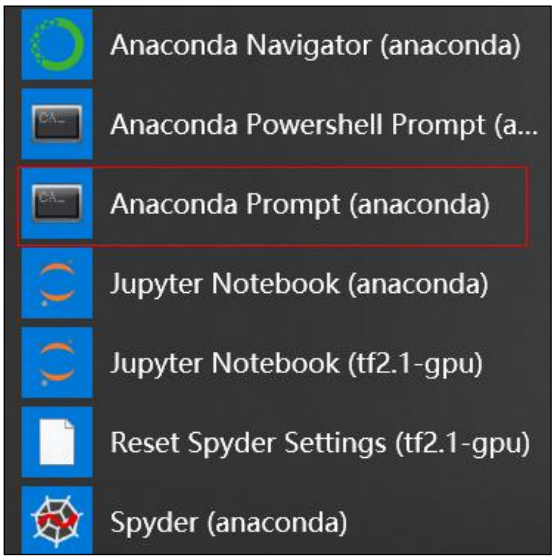


Figure 1-35

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

Step 2 Run the **pip install tensorflow-gpu==2.0.0** command, as shown in Figure 1-36.

Figure 1-36

The installation is successful if information shown in Figure 1-37 is displayed.

```
Successfully installed absl-py-0.9.0 astor-0.8.1 cachetools-4.0.0 chardet-3.0.4 gast-0.2.2 google-auth-1.11.2 google-auth-oauthlib-0.4.1 google-pasta-0.1.8 grpcio-1.27.2 h5py-2.10.0 idna-2.9 keras-applications-1.0.8 keras-preprocessing-1.1.0 markdown-3.2.1 numpy-1.18.1 oauthlib-3.1.0 opt-einsum-3.2.0 protobuf-3.11.3 pyasn1-0.4.8 pyasn1-modules-0.2.8 requests-2.23.0 requests-oauthlib-1.3.0 rsa-4.0 scipy-1.4.1 six-1.14.0 tensorboard-2.1.1 tensorflow-2.1.0 tensorflow-estimator-2.1.0 termcolor-1.1.0 urllib3-1.25.8 werkzeug-1.0.0 wrapt-1.12.0
```

Figure 1-37

Step 3 GPU acceleration can be implemented only when the local host has an independent graphics card and CUDA and cuDNN are installed. Install CUDA first. Download the CUDA version from https://developer.nvidia.com/cuda-10.0-download-archive?target_os=Windows&target_arch=x86_64&target_version=10&target_type=exelocal.

This URL is directed to CUDA 10.0. To download other versions, click **Legacy Releases**.

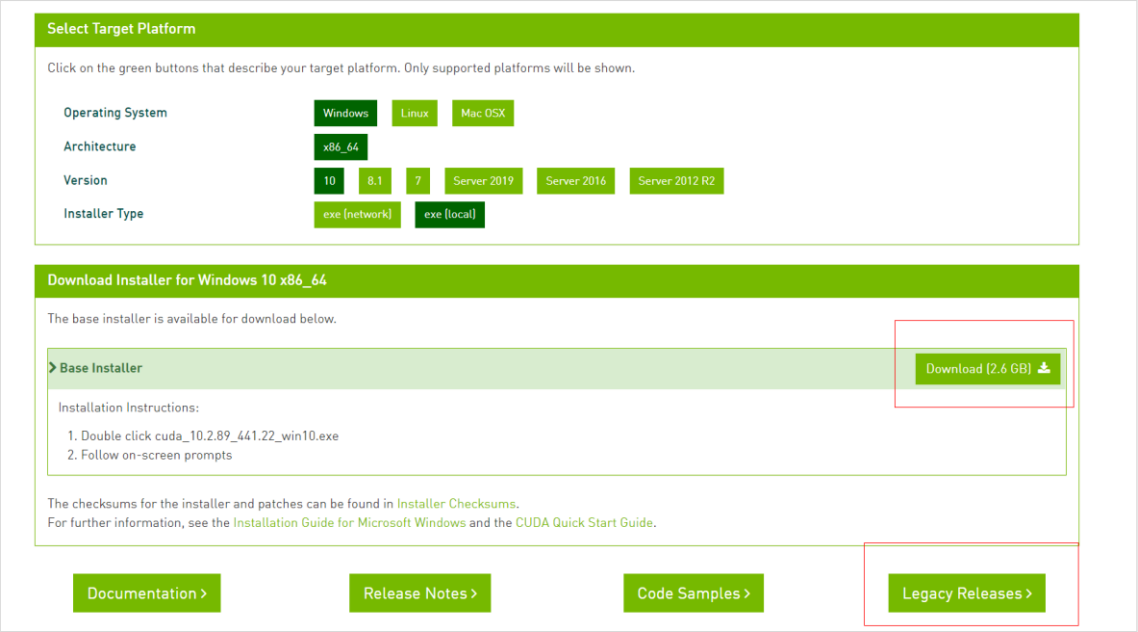


Figure 1-38

Step 4 Install CUDA. At the beginning, the program automatically decompresses the package to the default directory. Click **OK** to go to the next step.

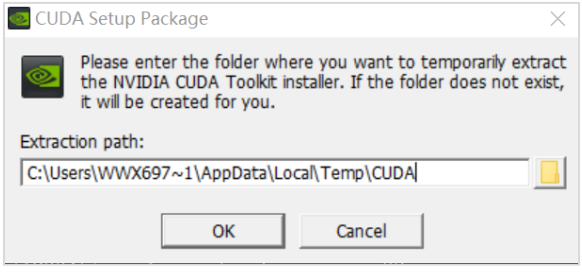


Figure 1-39

Step 5 Select **Custom** and click **NEXT**.

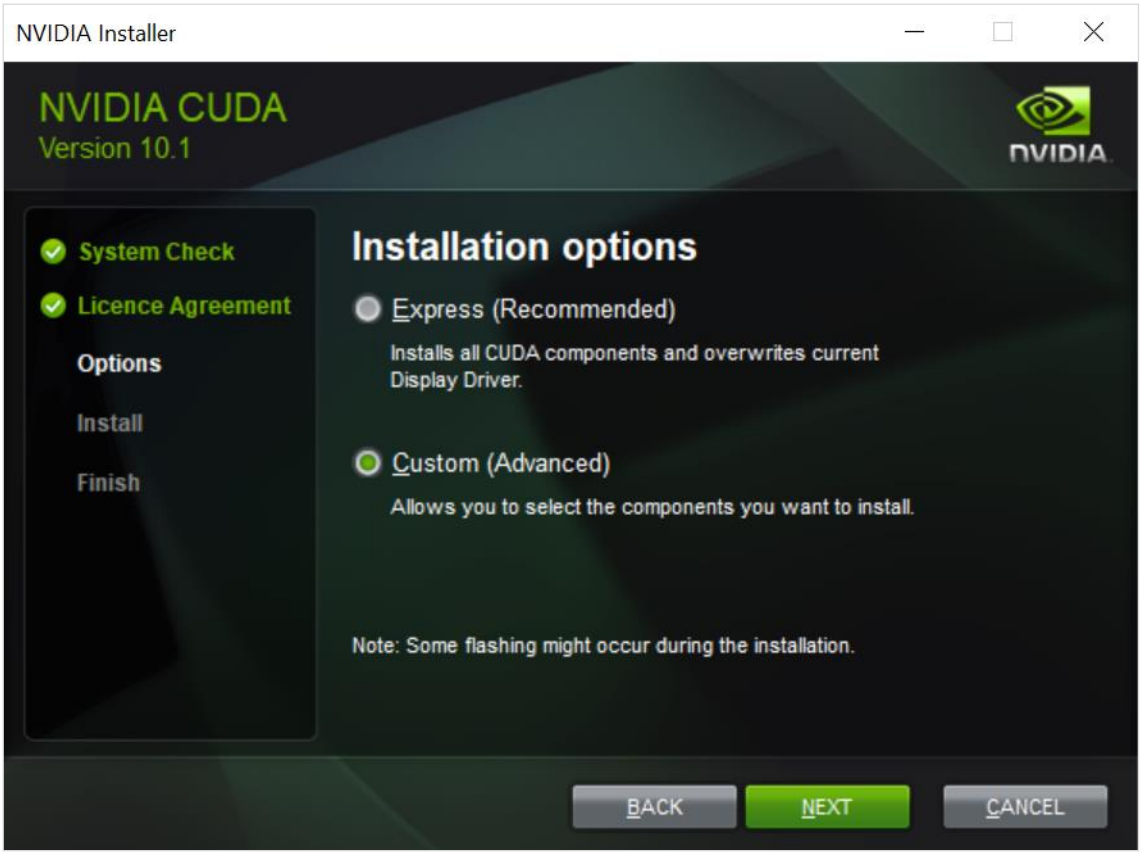


Figure 1-40

Wait until the installation is complete.

Step 6 Install cuDNN. Register with the official website (https://developer.nvidia.com/cudnn) and download the cuDNN software.

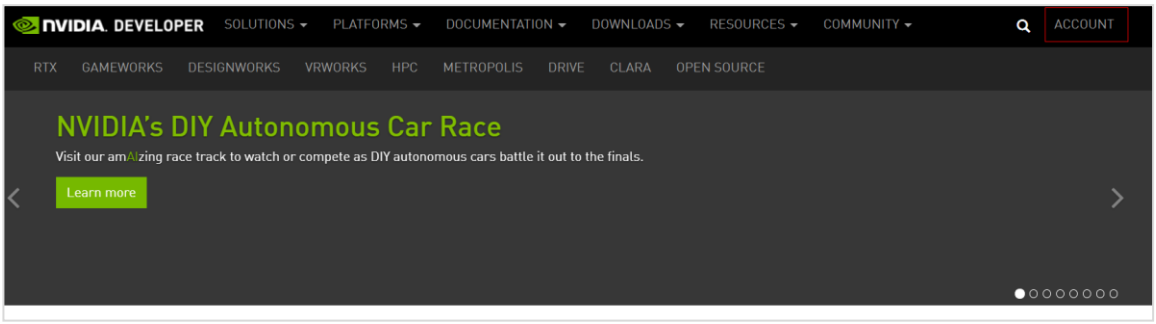


Figure 1-41

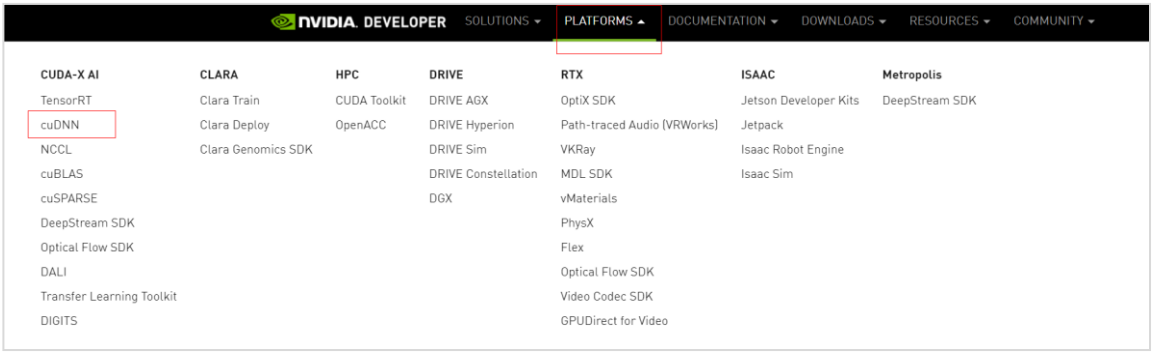


Figure 1-42

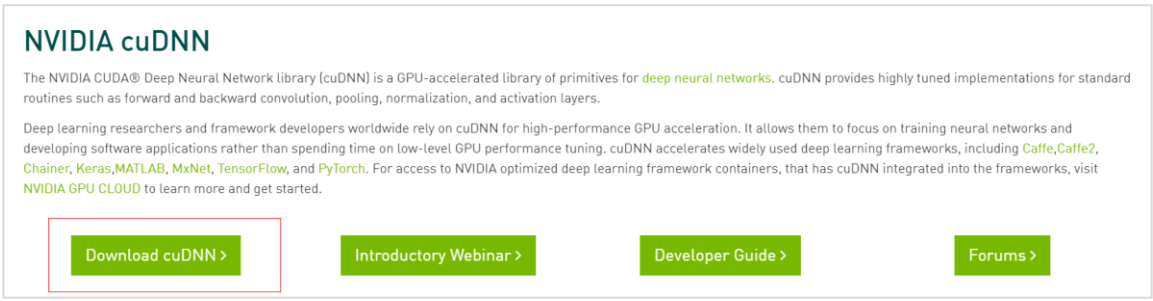


Figure 1-43

Step 7 The cuDNN version must match the CUDA version. Since CUDA 10.1 is installed, download cuDNN 7.6.5.

Figure 1-44

Step 8 Decompress the package downloaded.

Three folders are generated, as shown in Figure 1-45.

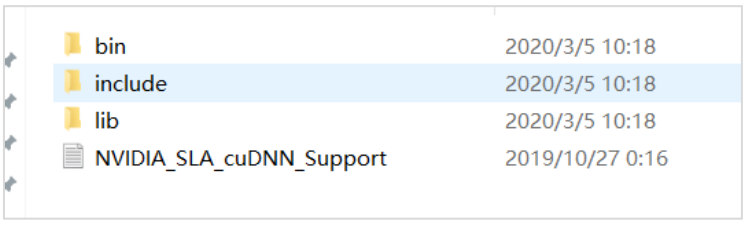


Figure 1-45

Step 9 Copy the three folders to the **CUDA** directory. If you do not change the paths, the default path is **C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0**. Replace the folders in the directory with the three folders.

Step 10 Configure CUDA environment variables. For details, see the operation of adding environment variables in section 1.3.4 "Installing TensorFlow 2.1.0 for GPU". Note that the version number must be 10.0.

Step 11 On the CLI, run **nvcc –V**.

If information similar to Figure 1-46 is displayed, the installation is successful.

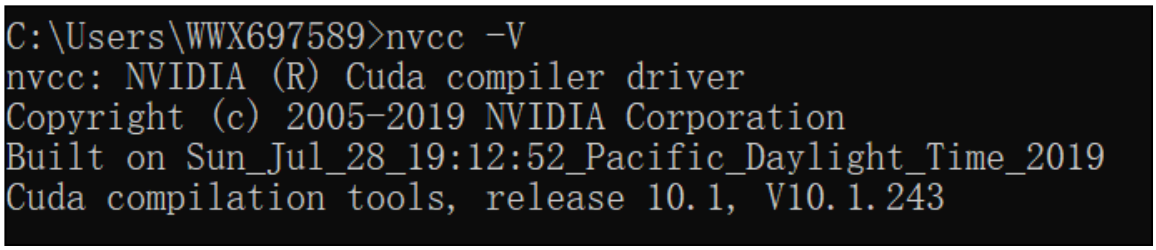


Figure 1-46

You can also run **import tensorflow** in the GPU environment and TensorFlow for GPU version. If no error is reported, the installation is successful.

1.4 Compiling Test Scripts in Real Time

1.4.1 Test Approach

TensorFlow introduces Keras in version 2.1.0. The test is to check whether Keras can be successfully imported and used.

1.4.2 Test Procedure

Step 1 Start Jupyter Notebook.

Start Jupyter Notebook of Anaconda from the start menu.

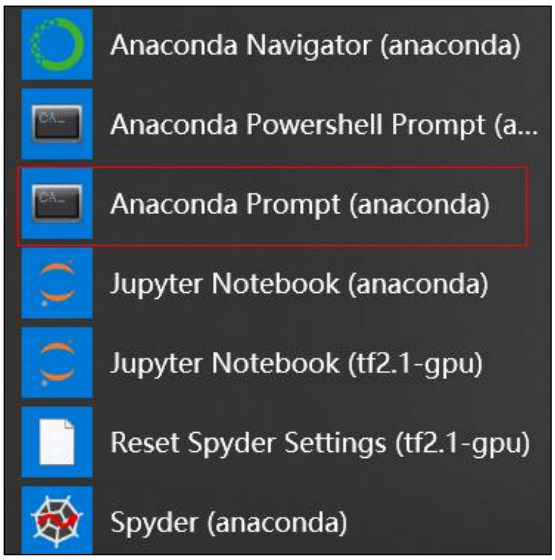


Figure 1-47

Step 2 Edit the script.

Click **New** on the right and select **Python 3**.

Figure 1-48

Enter the following content in the text box:

```
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)
```

Step 3 Start the test.

Click **Run** to run the Python script.

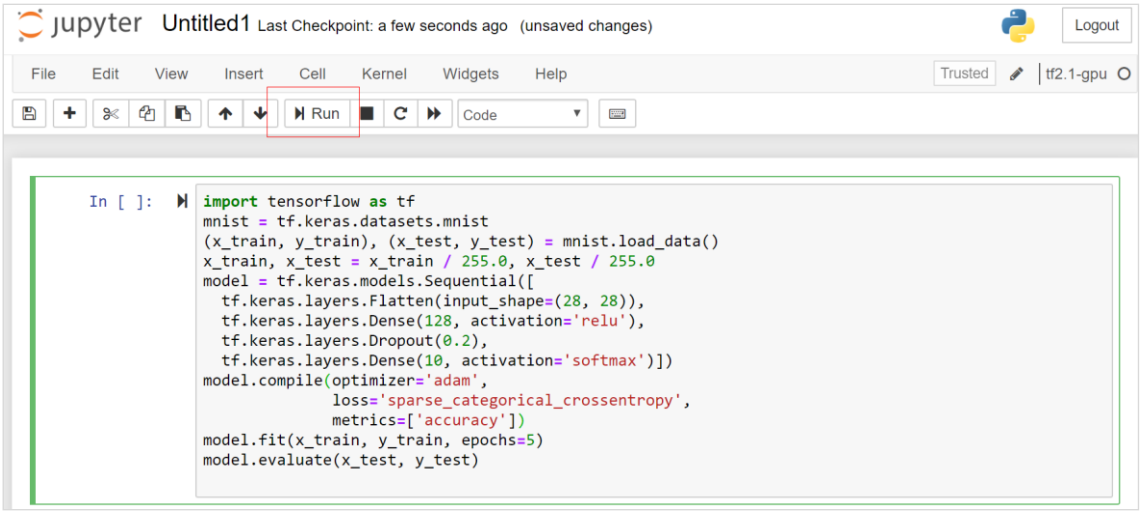


Figure 1-49

The output is as follows:

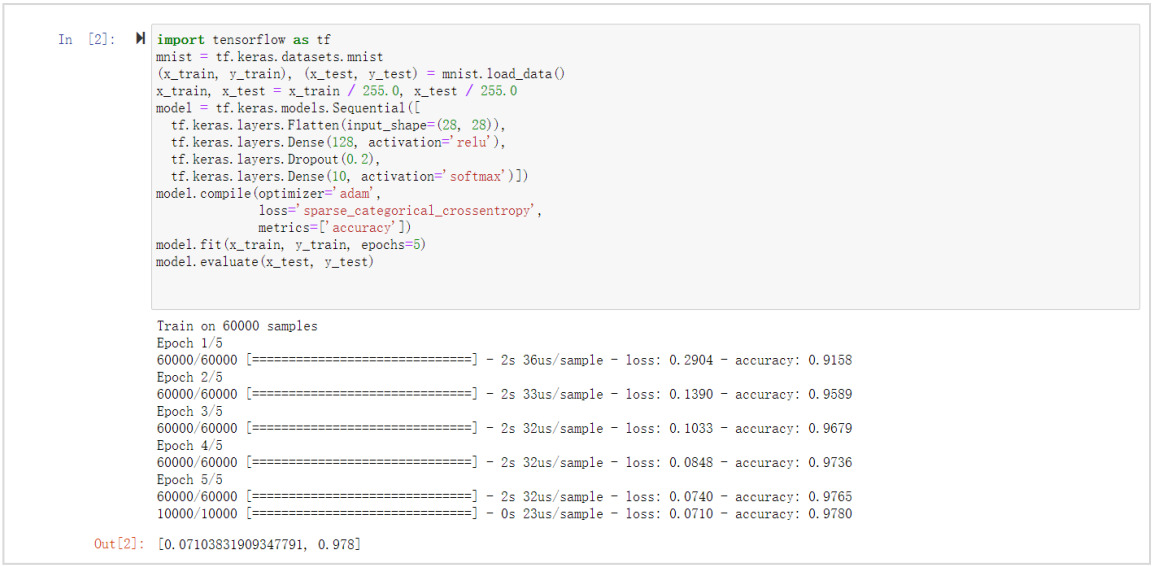


Figure 1-50

As shown in Figure 1-50, Keras is successfully imported and a basic neural network is built. After the mnist dataset is used for training, the test is complete. The test result indicates that TensorFlow 2.1.0 is successfully installed.

1.5 Anaconda Virtual Environments

1.5.1 Introduction to Virtual Environments

Anaconda is popular because of virtual environments. It allows multiple independent Python environments to be created on a host for developers to use. When the dependencies between different Python modules are disordered or different development framework applications exist, virtual environments keep these dependencies in separate sandboxes so users can switch between both applications easily and get them running.

Note: Modules in the virtual environments are independent of each other. If you have installed TensorFlow in environment A and want to use TensorFlow in environment B, you need to install TensorFlow in environment B. This also applies to Spyder and Jupyter Notebook.

1.5.2 Creating a Virtual Environment

Step 1 Start Anaconda Navigator.

Start Anaconda Navigator from the start menu.

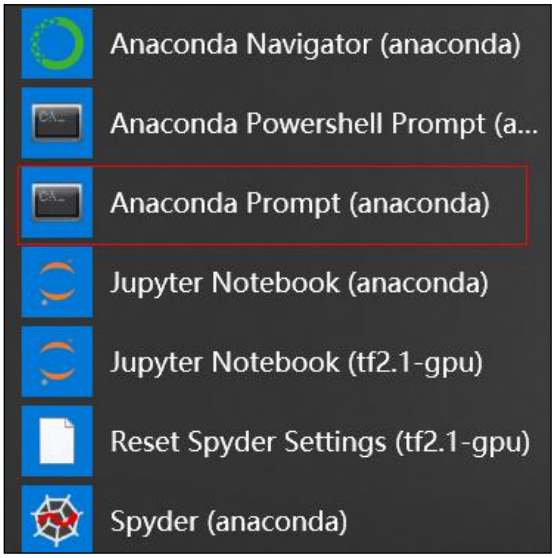


Figure 1-51

Step 2 Create a virtual environment.

Click **Environments** on the main menu, as shown in Figure 1-52.

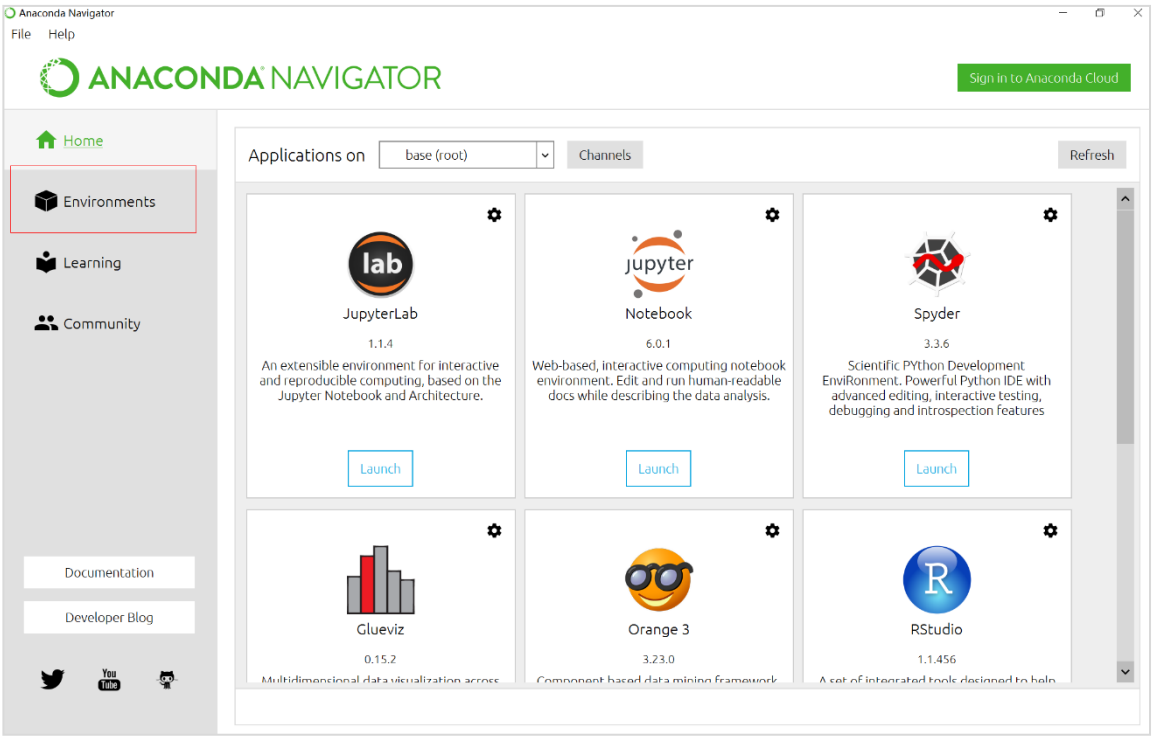


Figure 1-52

Click **Create**.

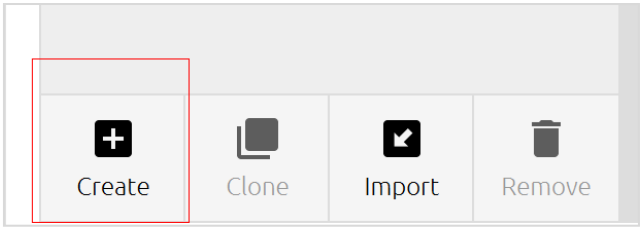


Figure 1-53

Step 3 In the dialog box that is displayed, set **Name**, and select **Python** and the version required by the development environment, for example, **3.6** or **3.7**.

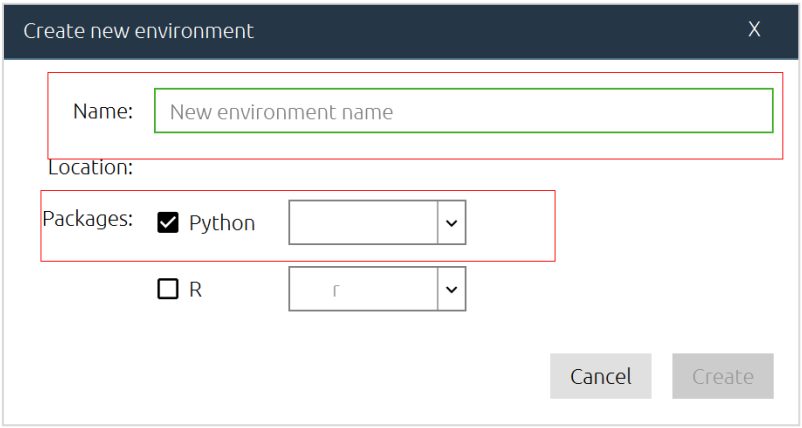


Figure 1-54

Click **Create**.

1.5.3 Creating a Virtual Environment Using the CLI

Step 1 Search for **cmd** in the search box on the taskbar and open the CLI.

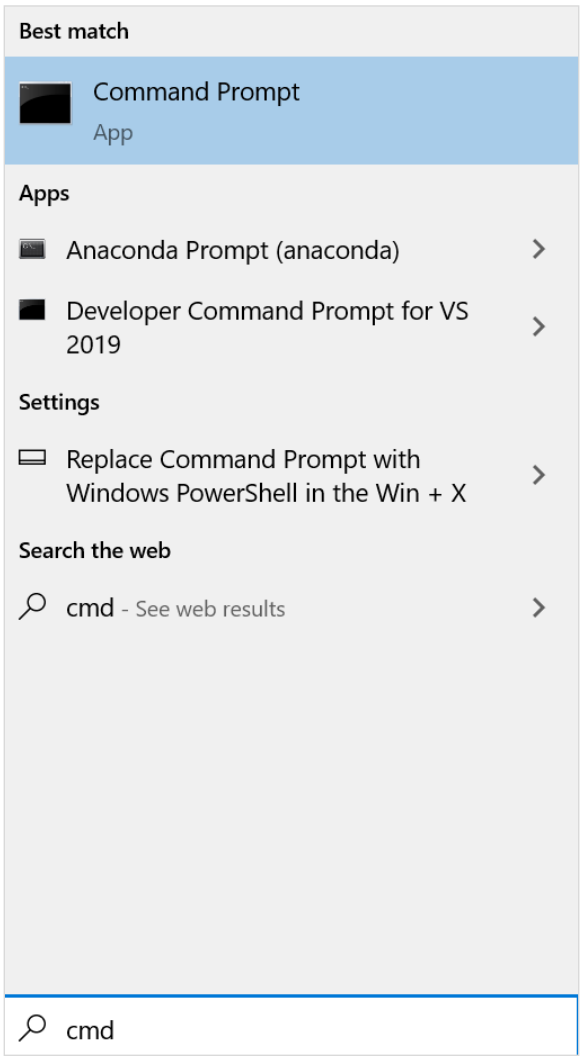


Figure 1-55

Step 2 Run the **conda create –n** *Environment name* **python==***x.x* command. Specify the environment name and Python version as required.

Figure 1-56

Step 3 When the information shown in Figure 1-57 is displayed, enter **y**.

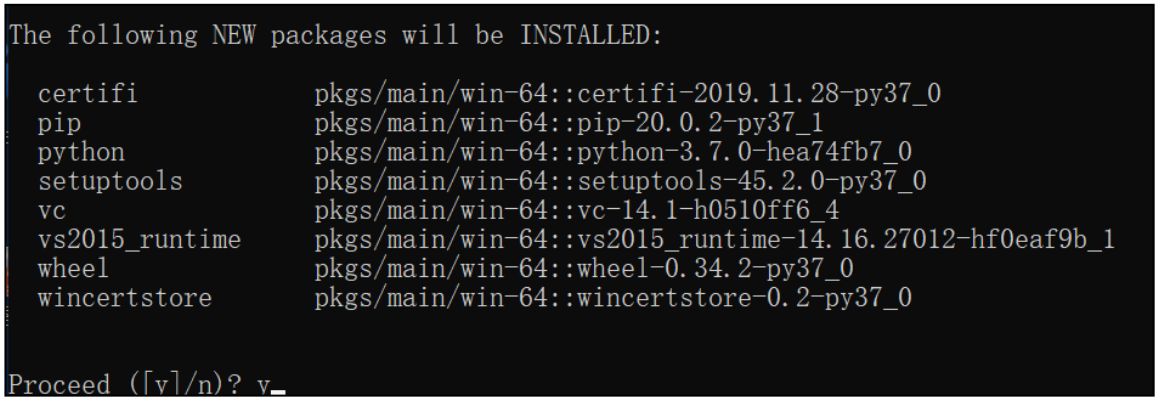


Figure 1-57

Wait until the installation is complete.

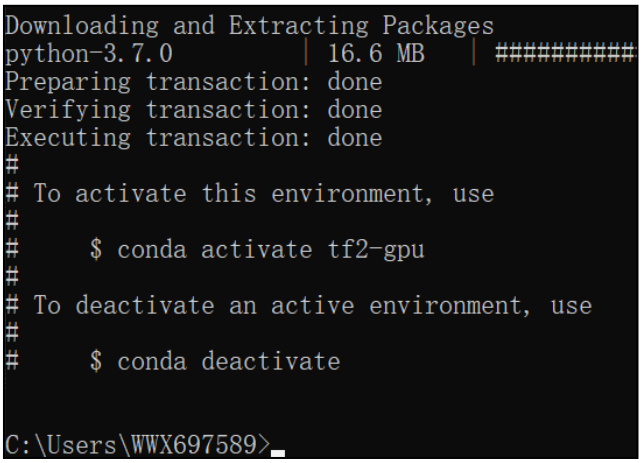


Figure 1-58

1.5.4 Activating a Virtual Environment

On the CLI, run the **activate *Environment name*** command to activate the specified virtual environment. If the name in the brackets before the command line is changed, the virtual environment is started and entered.

```
C:\Users\WWX697589>activate tf2-gpu

(tf2-gpu) C:\Users\WWX697589>
```

Figure 1-59

You can run the **pip** or **conda** command to install modules.

1.5.5 Viewing Virtual Environments

You can query created virtual environments on Anaconda Navigator.

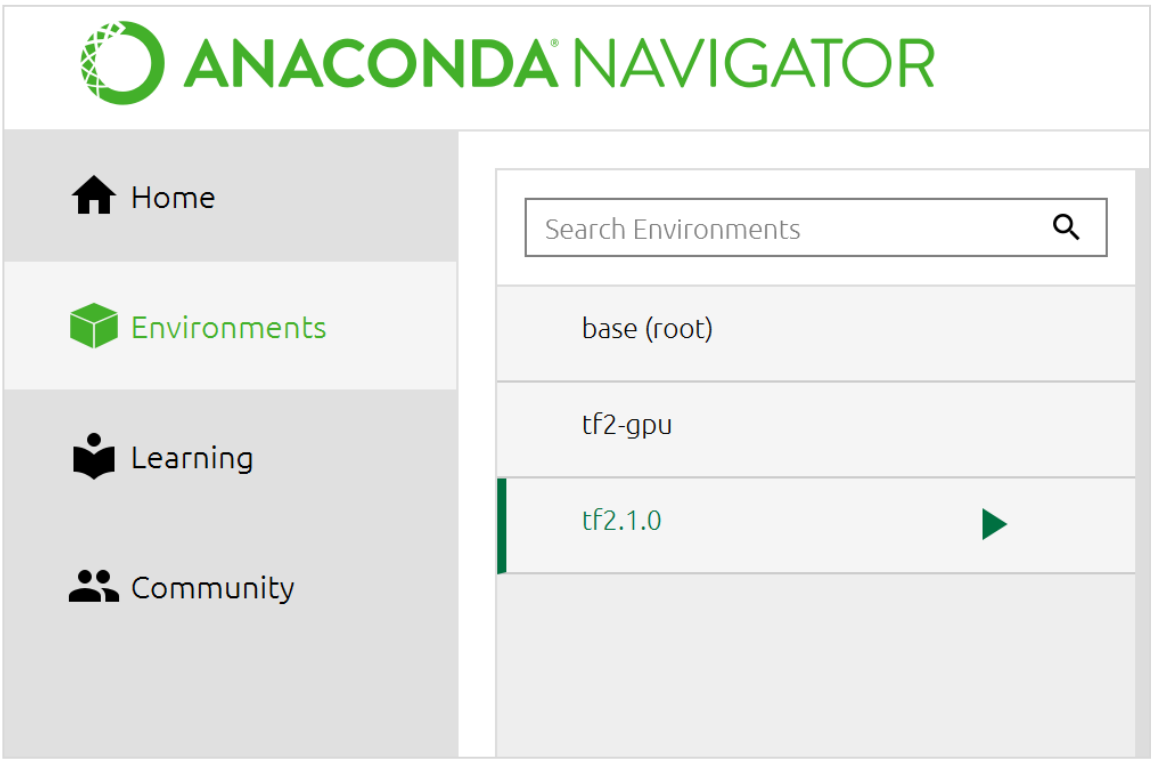


Figure 1-60

You can click the triangle on the right of the environment name to access the CLI of the virtual environment.

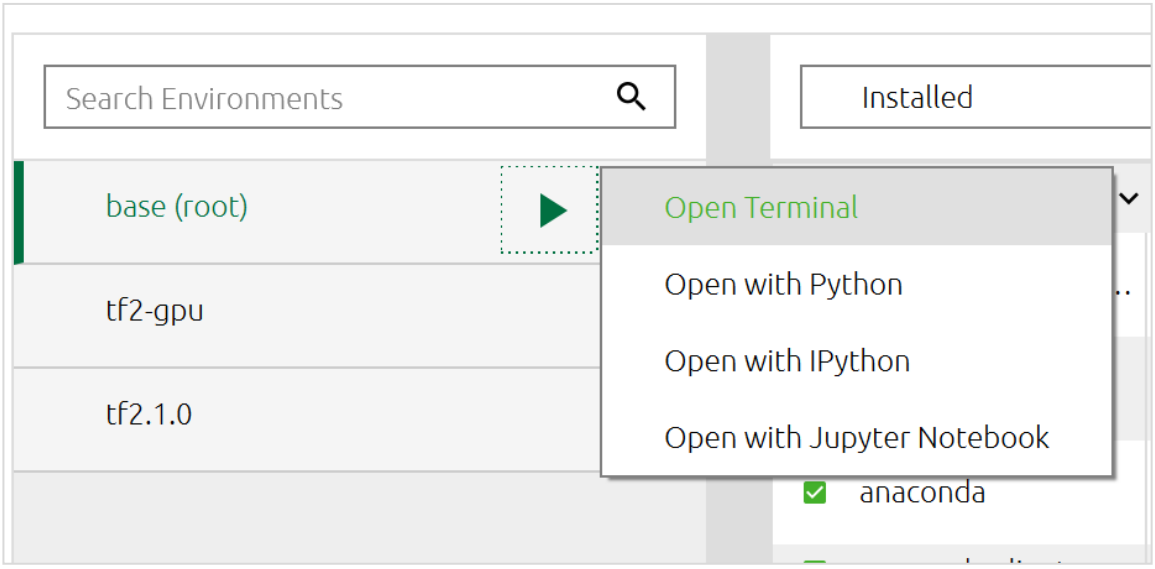


Figure 1-61

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

1.5.6 Deleting a Virtual Environment

Run the **conda remove –n** *Environment name* **–all** command and enter **y** to delete the environment.

```
C:\Users\WWX697589>conda remove -n test -all
usage: conda-script.py [-h] [-V] command ...
conda-script.py: error: unrecognized arguments: -all

C:\Users\WWX697589>conda remove -n test --all

Remove all packages in environment D:\anaconda\envs\test:


## Package Plan ##

  environment location: D:\anaconda\envs\test


The following packages will be REMOVED:

  certifi-2019.11.28-py37_0
  pip-20.0.2-py37_1
  python-3.7.0-hea74fb7_0
  setuptools-45.2.0-py37_0
  vc-14.1-h0510ff6_4
  vs2015_runtime-14.16.27012-hf0eaf9b_1
  wheel-0.34.2-py37_0
  wincertstore-0.2-py37_0


Proceed ([y]/n)?
```

Figure 1-62

2 Configuring the macOS Experiment Environment

2.1 Introduction

2.1.1 About This Experiment

In this experiment, you need to set up a development environment for all HCIA-AI experiments based on macOS. You need to download and install Anaconda, select the Python version, install dependencies, and install Jupyter Notebook.

2.1.2 Objectives

Set up an HCIA-AI experiment environment of the CPU version based on macOS.

2.1.3 Modules Required

- Anaconda 3.7
- Python 3.7
- TensorFlow 2.1.0 or 2.0.0

2.2 Downloading Anaconda and Configuring the Python Environment

Anaconda is a distribution of Python for scientific computing. It supports Linux, macOS, and Windows. It provides simplified package management and environment management, and easily deals with the installation issues when the system has multiple Python versions and third-party packages. Anaconda uses the Conda command to manage packages and environments. It contains Python and related tools. Anaconda is a Python tool for enterprise-level big data analytics. It contains more than 720 open-source data-science packages, including data visualization, machine learning, and deep learning. It can be used for data analysis, big data and AI fields.

After installing Anaconda, you do not need to install Python.

2.2.1 Downloading Anaconda

Step 1 Visit https://www.anaconda.com/, click **Download**, and download the macOS version, as shown in Figure 2-1.

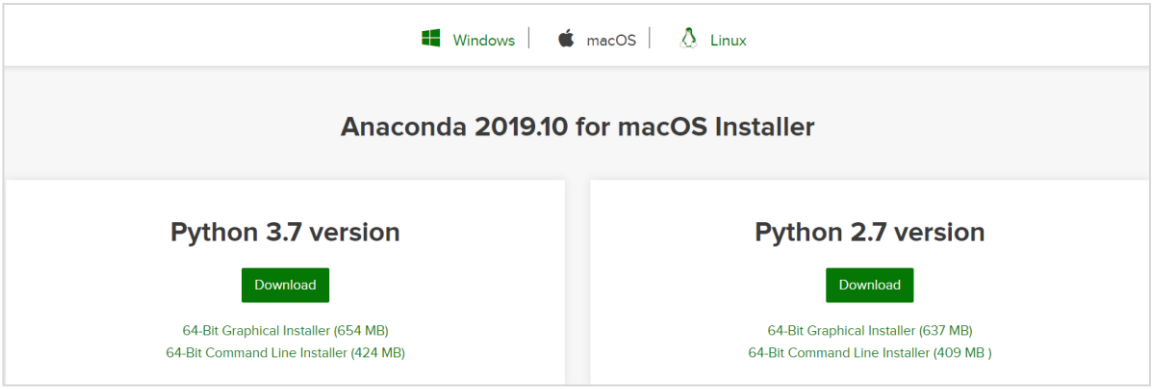


Figure 2-1

Download the Python 3.7 installation package of 654 MB.

2.2.2 Installing Anaconda

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

Step 2 Click **Continue** and **Install** buttons repeatedly.

Step 3 Enter the system password.

Wait until the installation is complete.

2.2.3 Creating a Python Virtual Environment

Step 1 Start Anaconda Navigator.

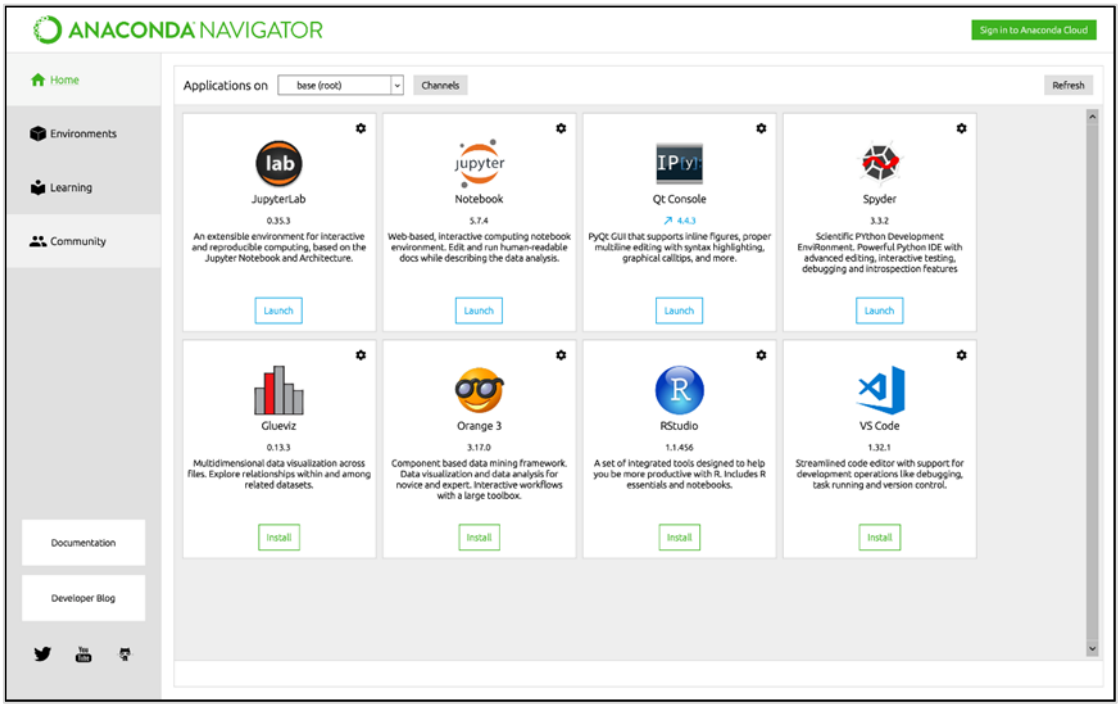


Figure 2-2

Step 2 Click **Environments** on the left. On the **Environments** page that is displayed, click **Create** in the lower left corner.

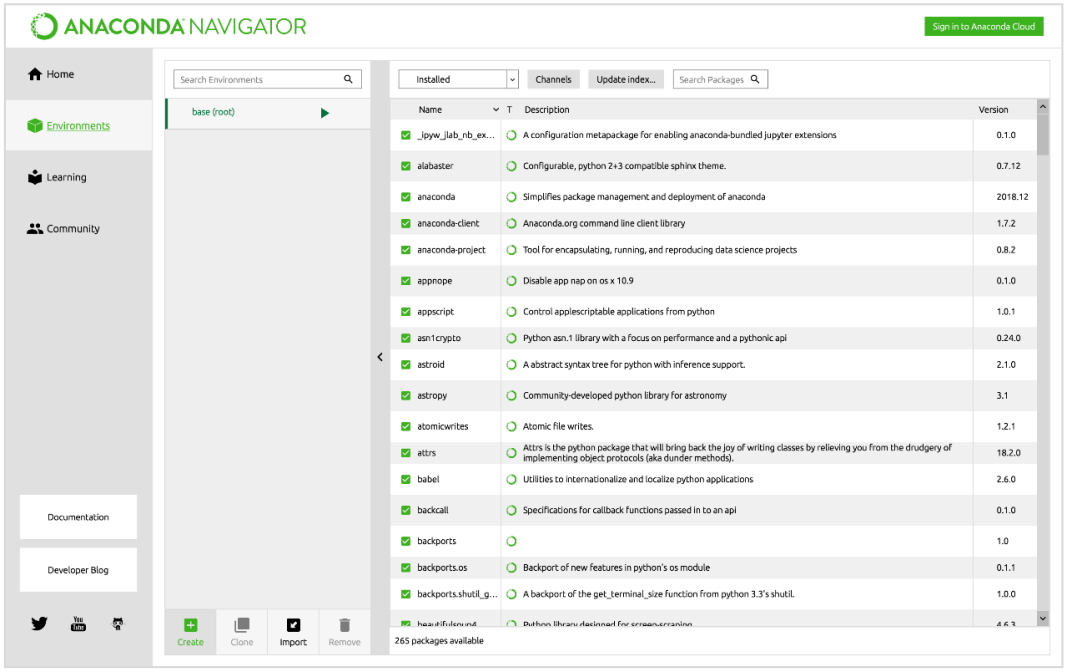
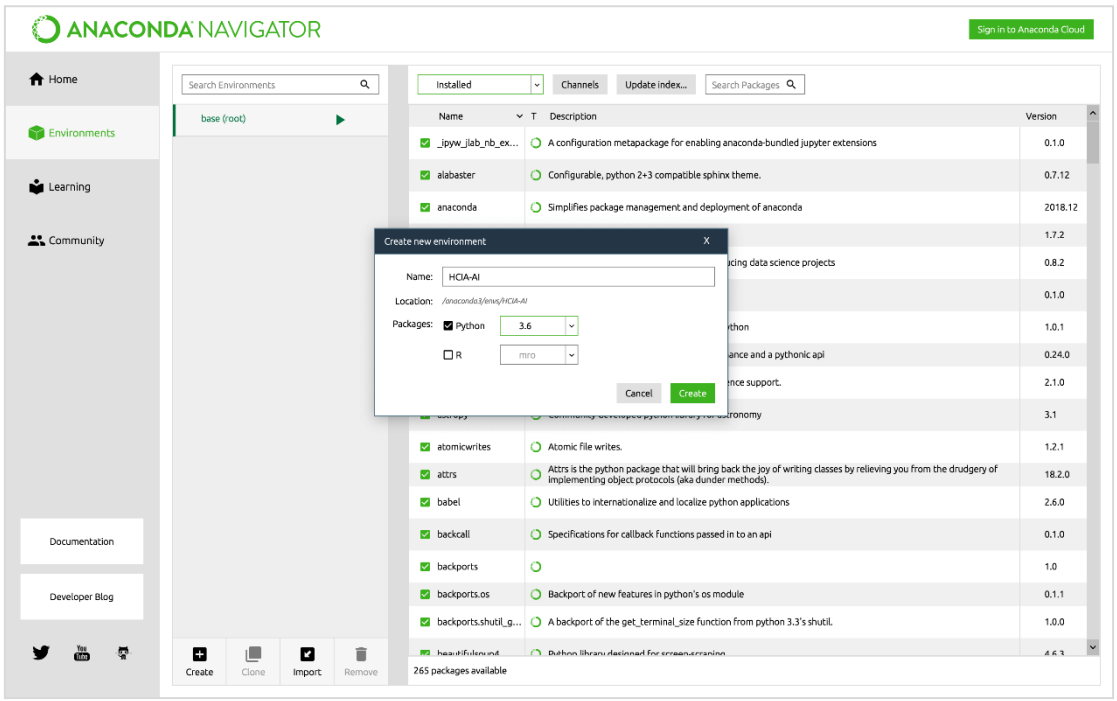


Figure 2-3

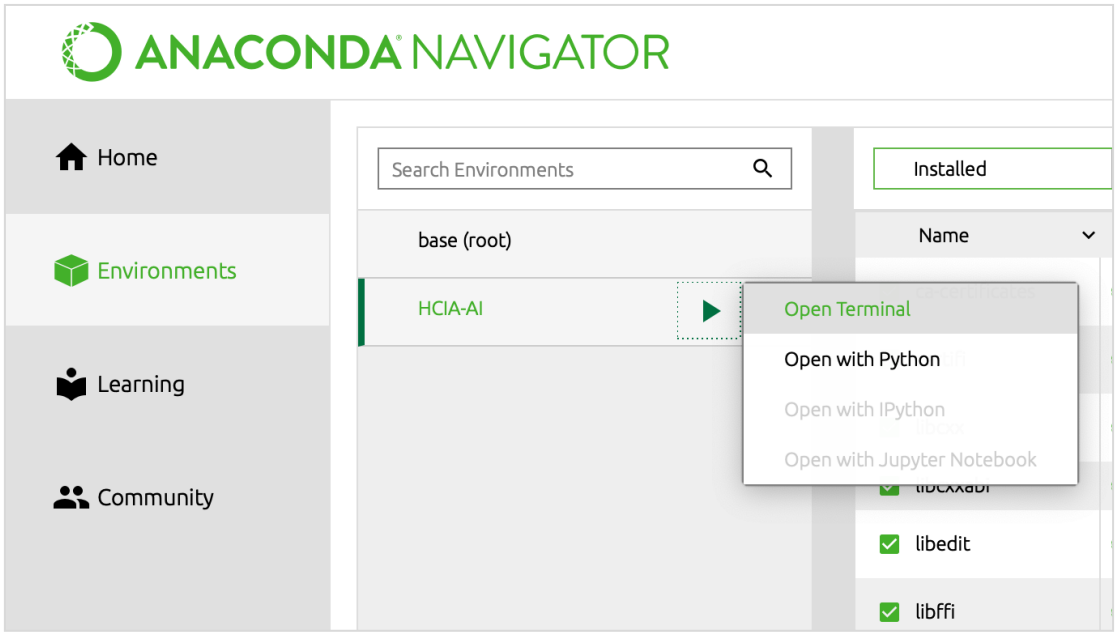
Step 3 Set the environment name, and select **Python** and its version. Version 3.6 is recommended.



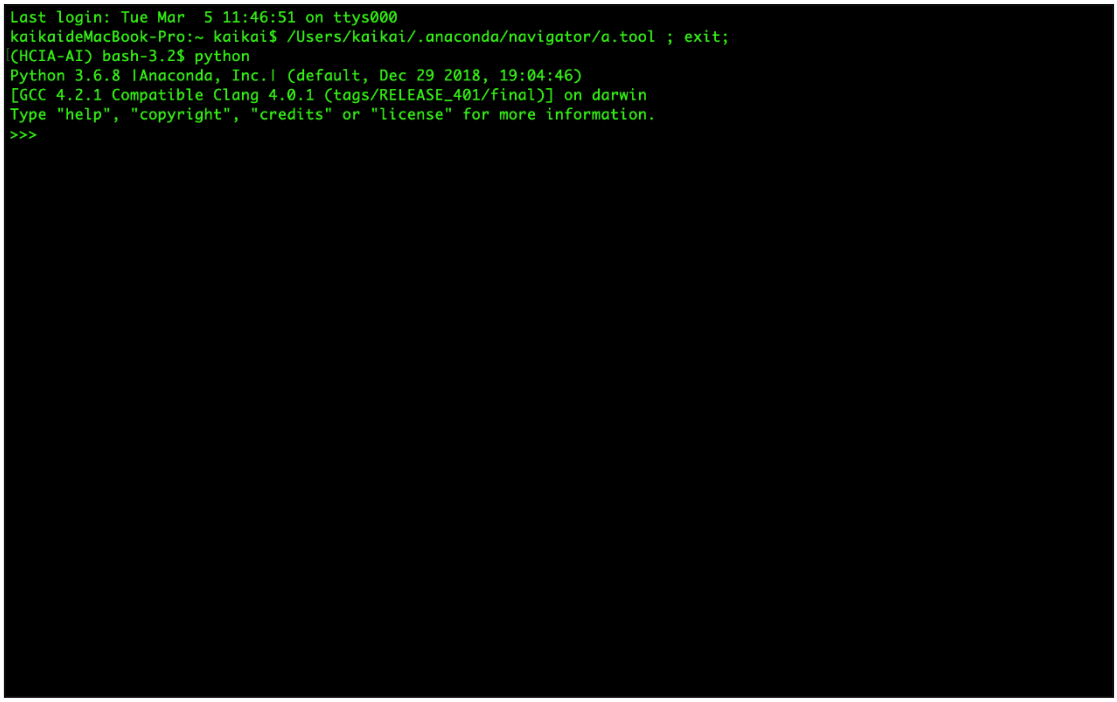
Wait until the environment is created successfully.

2.2.4 Testing the Environment

Step 1 Click the triangle next to the environment name to access the newly created environment and select **Open Terminal**.



Step 2 Enter **python** to check the Python version in the environment.



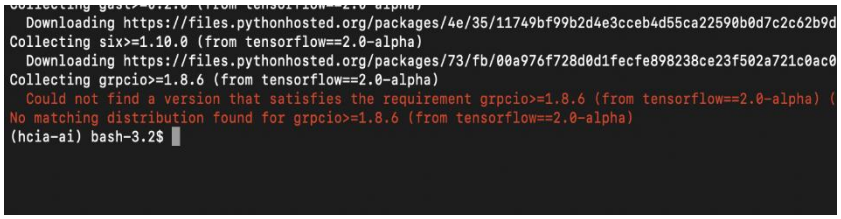
"Python 3.6.8" is displayed, which indicates that the installation is successful.

2.3 Installing TensorFlow

2.3.1 Installing TensorFlow 2.1.0

Step 1 Run the **pip install tensorflow-cpu** command to install the TensorFlow2.1.0 framework.

Step 2 If the following information is displayed, there are package missing. Manually install the corresponding packages.



Step 3 After the packages are installed, run **pip install tensorflow==2.1.0**, and wait until the installation is complete.

2.3.2 Installing TensorFlow 2.0.0

Run the **pip install tensorflow==2.0.0** command to install TensorFlow 2.0.0. For details, see "Installing TensorFlow 2.1.0".

2.4 Installing Jupyter Notebook

Jupyter Notebook can be installed by using the Terminal or Anaconda Navigator.

2.4.1 Installing Jupyter Notebook Using the Terminal

Run the **pip install jupyter notebook** command on the Terminal.

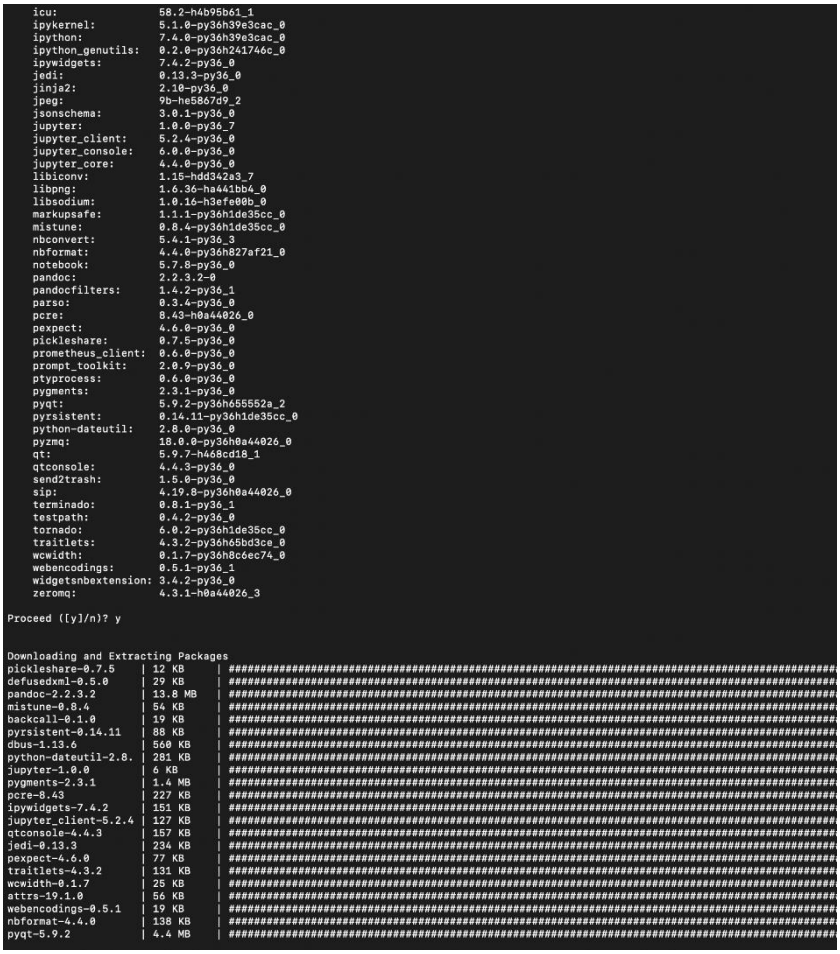


Figure 2-4

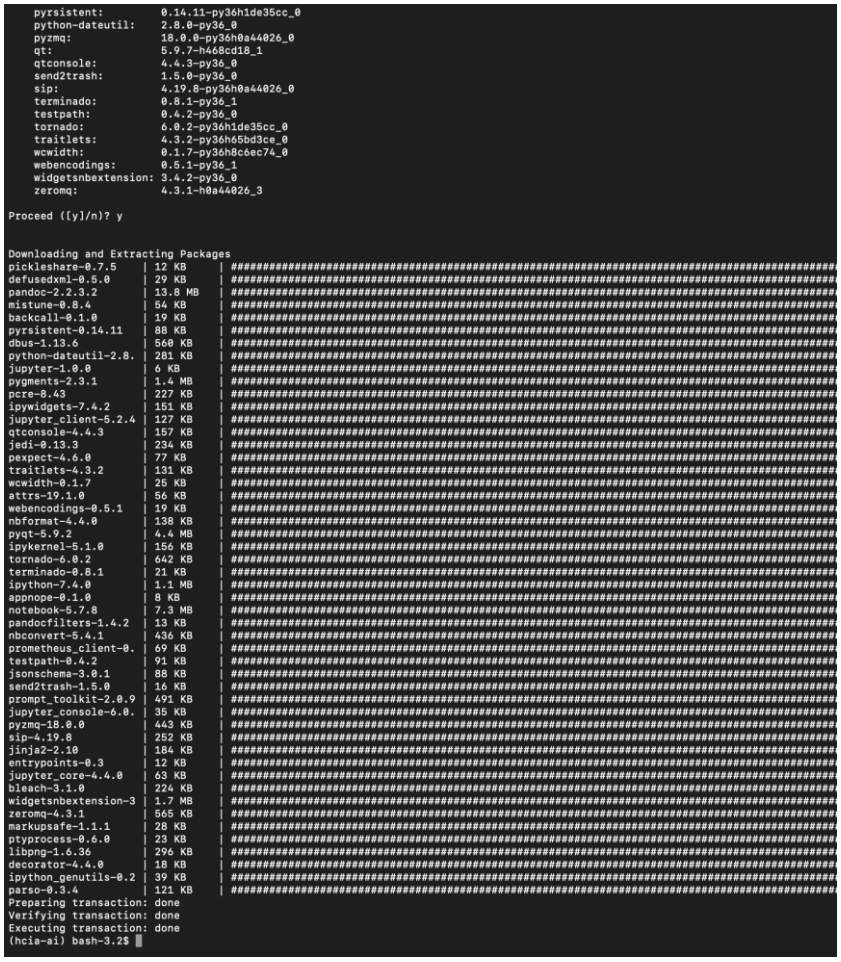


Figure 2-5

Wait until the installation is complete.

2.4.2 Installing Jupyter Notebook Using Anaconda Navigator

Step 1 Start Anaconda.

On the Anaconda home page, click **Home** on the left.

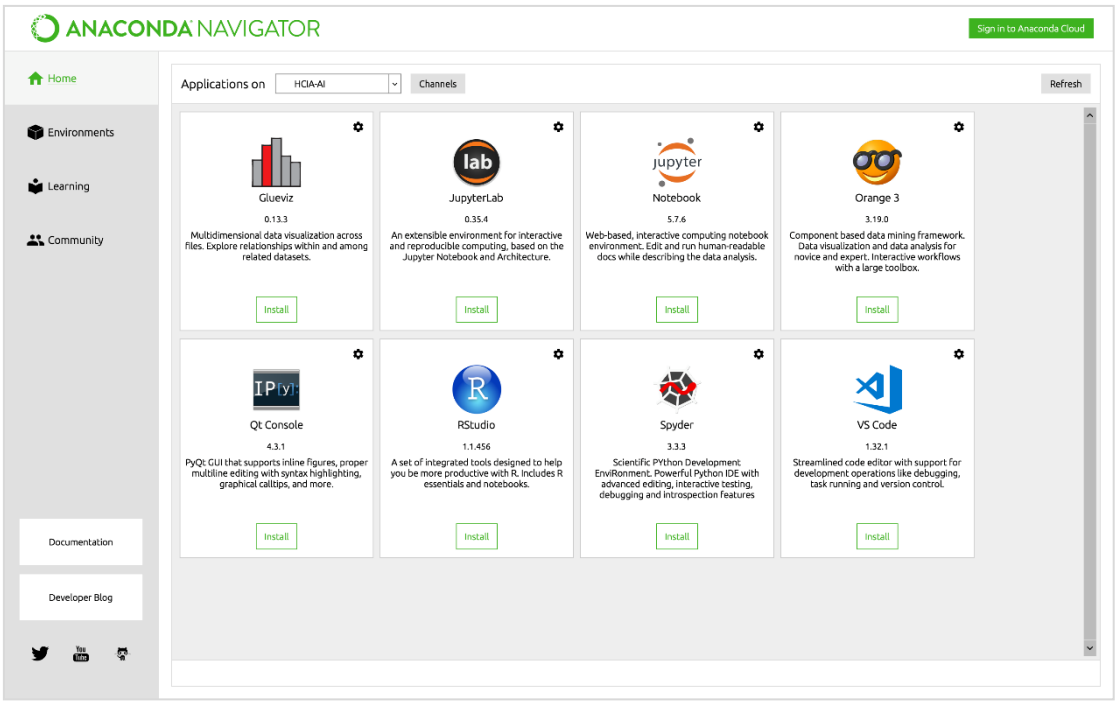


Figure 2-6

Step 2 Install Jupyter Notebook.

Set **Application on** to **HCIA-AI** and click **Install** under **Jupyter Notebook** on the right. After the installation is complete, the page shown in Figure 2-7 is displayed.

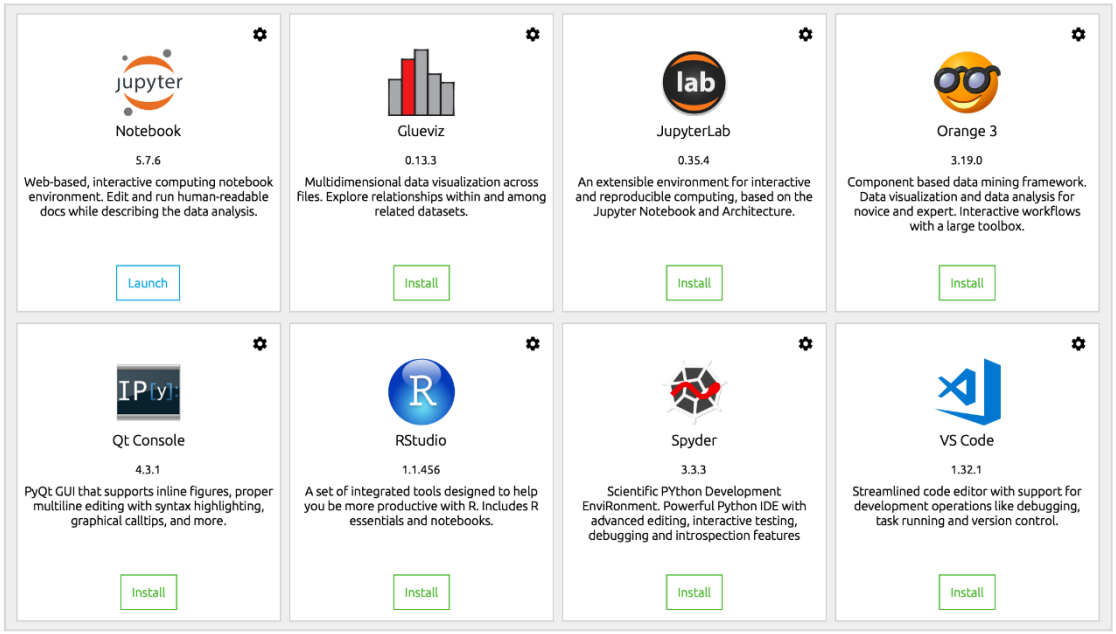


Figure 2-7

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

Step 3 Verify the installation.

Click **Launch** under **Jupyter Notebook**. The Jupyter home page is displayed.

Figure 2-8

Click **New** in the upper right corner of the page and select **Python 3** to create a Jupyter file.

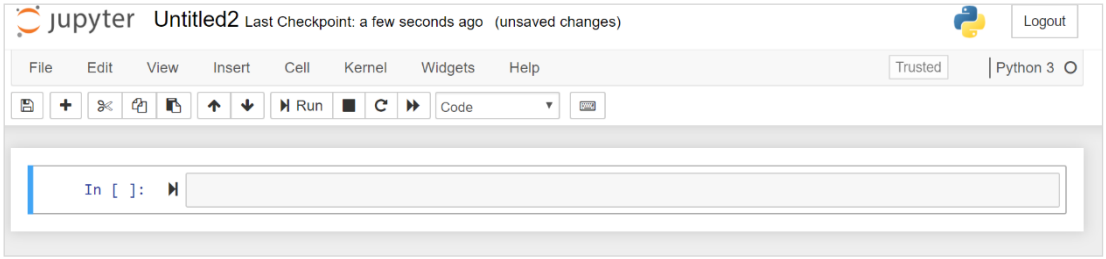


Figure 2-9

Enter the following content in the text box:

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

Click **Run**, as shown in Figure 2-10.

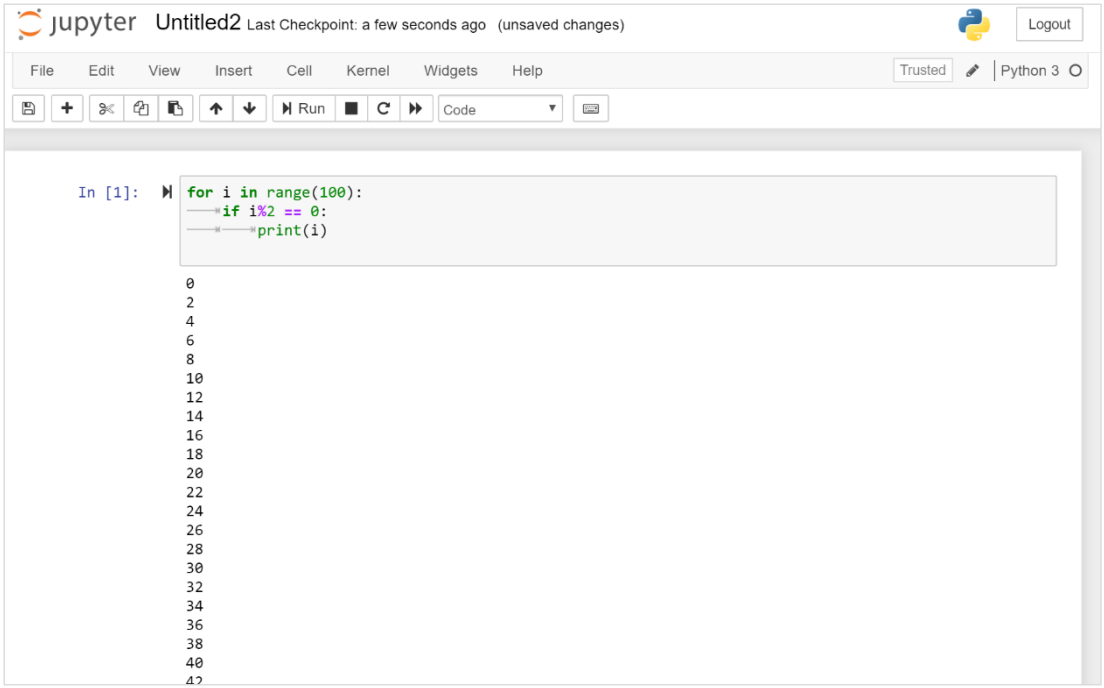


Figure 2-10

2.5 Performing Test Cases

Verify that Python 3.6 and TensorFlow development environments created based on Anaconda can run properly.

To be more specific, verify that the graph mechanism can be used to perform constant additions based on Python 3.6, Jupyter Notebook, and TensorFlow frameworks.

2.5.1 Test Case

Step 1 Start Jupyter.

Click **Launch**.

Figure 2-11

Step 2 Create a Jupyter project.

Click **New** in the upper right corner and select **Python 3** to create a script.

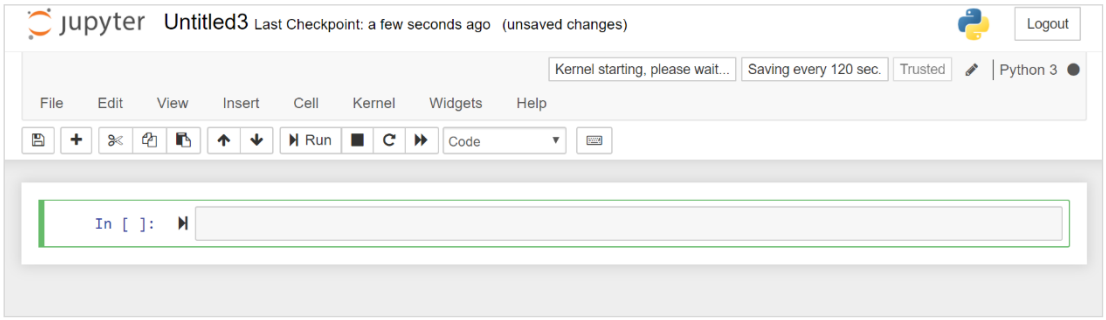


Figure 2-12

Click **Untitled3** in the upper left corner and name the script **tf_demo**.

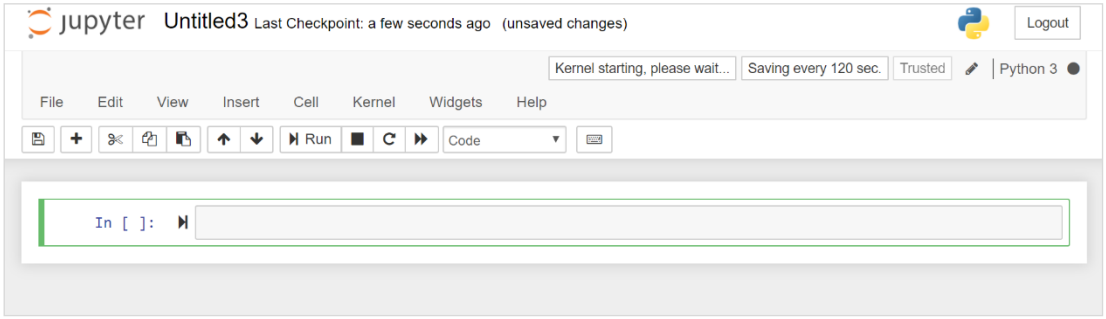


Figure 2-13

Enter the following code:

```
# 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 operation.
mul = tf.matmul(mat1,mat2)

print(mul)
```

Result:

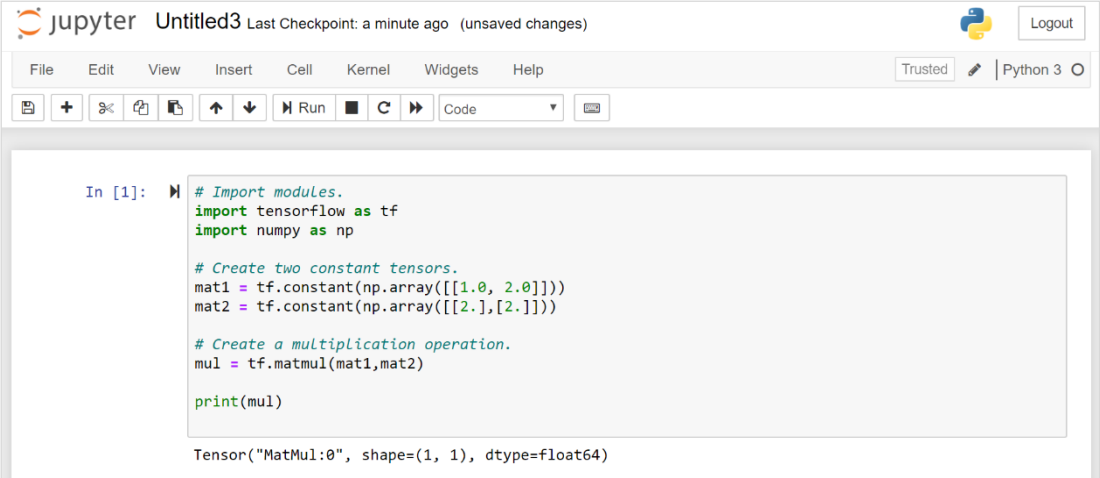


Figure 2-14

Step 3 Verify Keras.

Verify the Keras module, which is a new feature introduced in TensorFlow 2.0.

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. The information shown in Figure 2-15 is displayed.

```
In [4]: 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

        Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
        11493376/11490434 [==============================] - 16s 1us/step

In [9]: 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'),
        ])

In [19]: model.compile(optimizer='adam',
                       loss=['sparse_categorical_crossentropy'],
                       metrics=['accuracy'],)
         model.fit(x_train,y_train,epochs=5)

         Epoch 1/5
         60000/60000 [==============================] - 2s 37us/sample - loss: 0.2982 - accuracy: 0.9131
         Epoch 2/5
         60000/60000 [==============================] - 2s 33us/sample - loss: 0.1423 - accuracy: 0.9582
         Epoch 3/5
         60000/60000 [==============================] - 2s 33us/sample - loss: 0.1093 - accuracy: 0.9667
         Epoch 4/5
         60000/60000 [==============================] - 2s 33us/sample - loss: 0.0893 - accuracy: 0.9722
         Epoch 5/5
         60000/60000 [==============================] - 2s 33us/sample - loss: 0.0758 - accuracy: 0.9758

Out[19]: <tensorflow.python.keras.callbacks.History at 0x10971f898>

In [20]: model.evaluate(x_test,y_test)

         10000/10000 [==============================] - 0s 24us/sample - loss: 0.0814 - accuracy: 0.9743

Out[20]: [0.08144361303178593, 0.9743]

In [ ]:
```

Figure 2-15

This experiment verifies whether the HCIA-AI experiment environment is successfully set up. If no error is reported during code running, the experiment environment is successfully set up.

3 Using Huawei Cloud

3.1 Overview

Huawei Cloud provides enriched cloud resources and services to meet the requirements of developers. The following describes how to apply for Huawei Cloud resources, including registering an account, performing real-name authentication, and obtaining AK and SK, and describes major Huawei Cloud services.

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

- Understand the AI development environment.
- Apply for an AI development environment.
- Perform functional tests in the environment you have set up.

3.2 Registering a Huawei Cloud Account

3.2.1 Account

The account is used to:

- Enable services required for the experiments on the Huawei Cloud website.
- Experience various Huawei Cloud services, including deep learning, machine learning, image recognition, speech recognition, and natural language processing.

3.2.2 Registration Procedure

Step 1 Visit the Huawei Cloud website (https://www.huaweicloud.com/intl/en-us/).

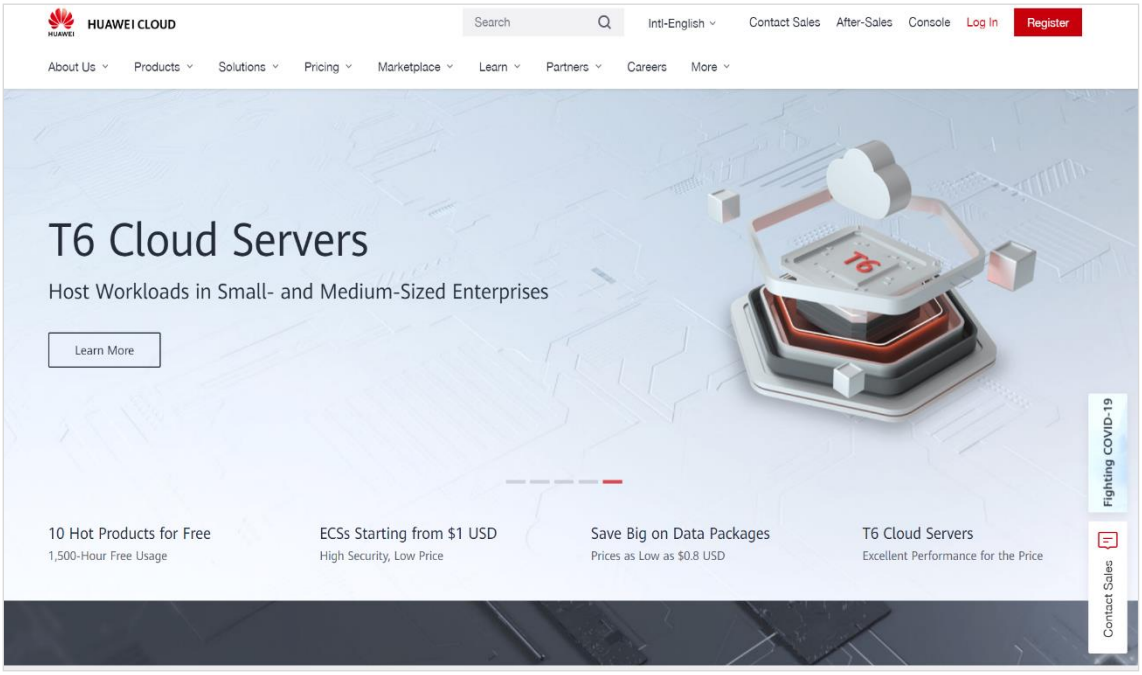


Figure 3-1

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

On the page displayed, enter user information.

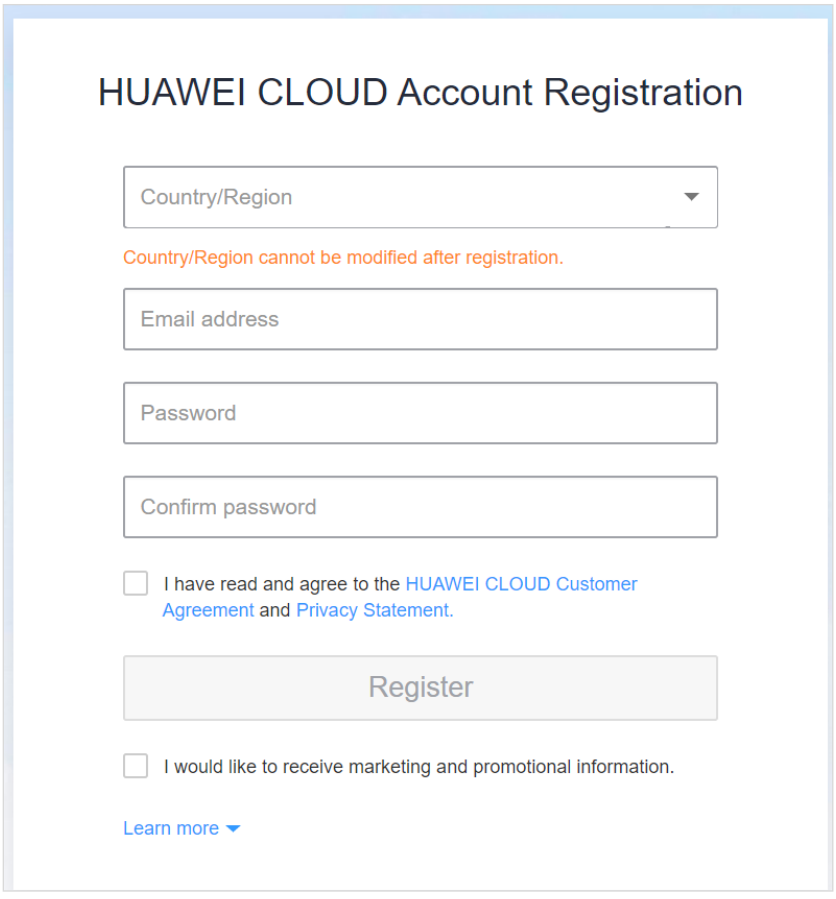


Figure 3-2

Step 3 Perform real-name authentication.

Choose **Real-Name Authentication** > **Individual**.

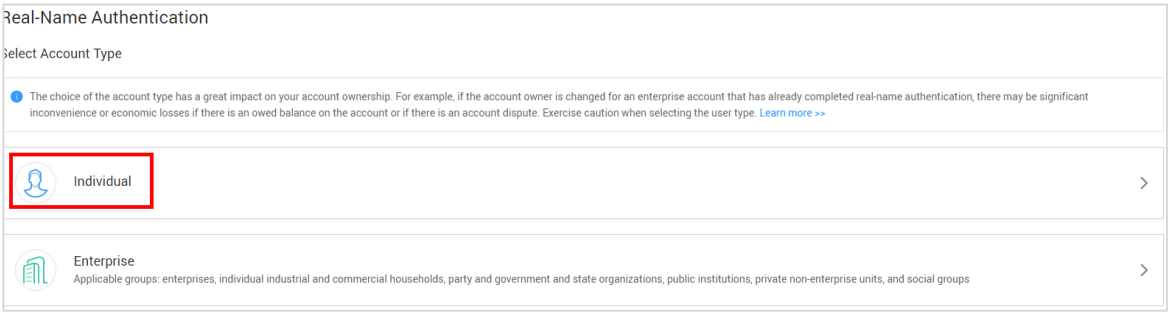


Figure 3-3

Choose **Individual Bank Card Authentication**.

Enter related information and submit the information.

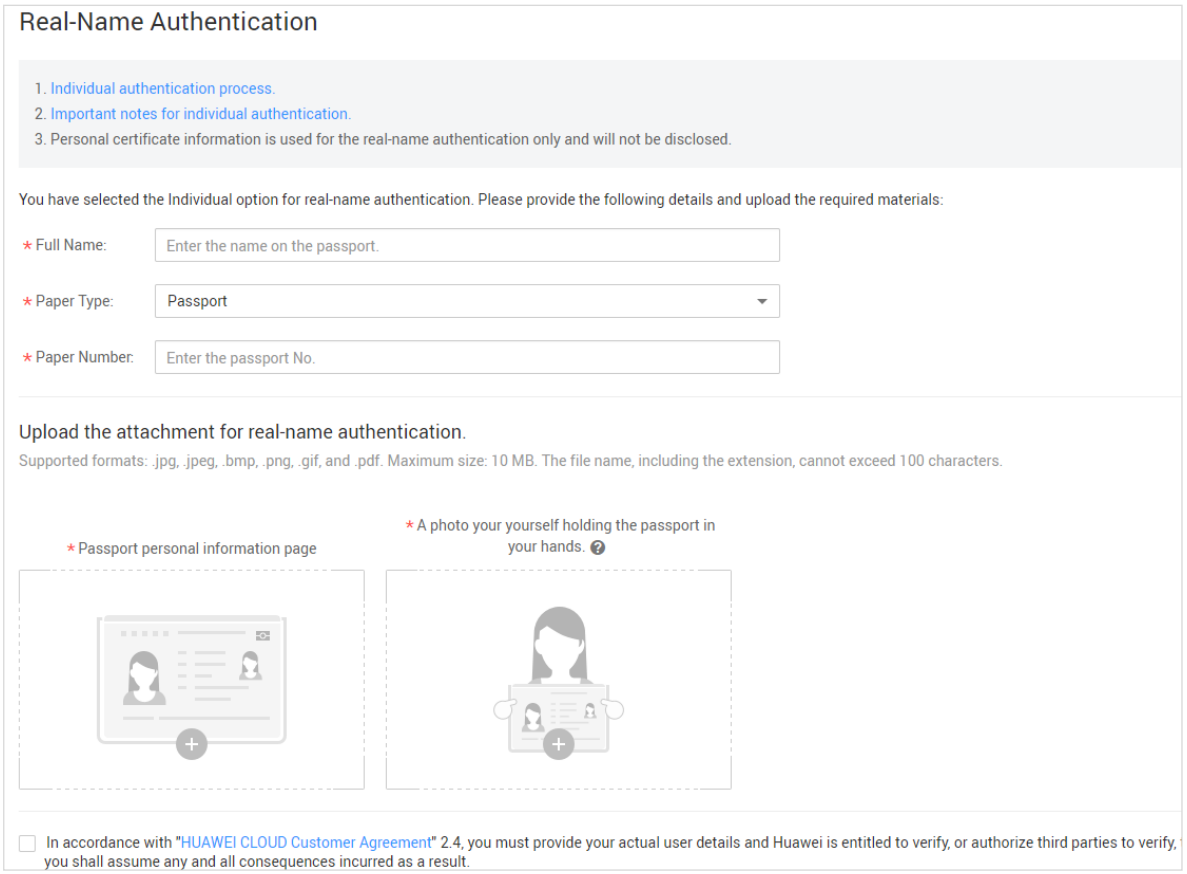


Figure 3-4

The real-name authentication is complete.

3.2.3 Approach for Applying for Services

- Huawei Cloud provides a large number of cloud services. You can apply for cloud resources based on your tasks and requirements.
- You can apply for cloud services and set up an environment as promoted on the corresponding pages.

3.3 Obtaining the Access Key and Secret Access Key

3.3.1 Overview

The access key (AK) and Secret access key (SK) are identity codes required for using services on Huawei Cloud. When using some services on Huawei Cloud for the first time, you are required to enter the AK and SK, as shown in Figure 3-5.

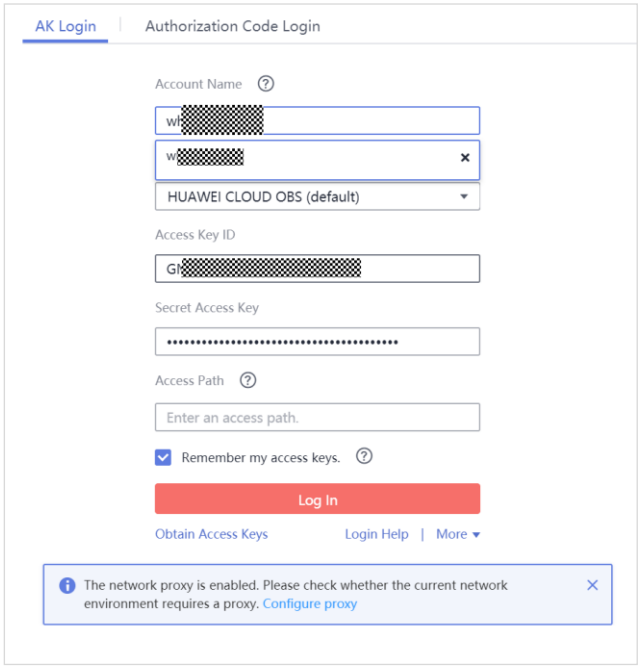


Figure 3-5

In Figure 3-5, **Access Key ID** is the AK, and **Secret Access Key** is the SK. You can use the OBS service only after entering the correct AK and SK.

3.3.2 Generating the AK and SK

Step 1 Log in to the Huawei Cloud website using the account registered, and click **My Account** in the upper right corner of the home page.

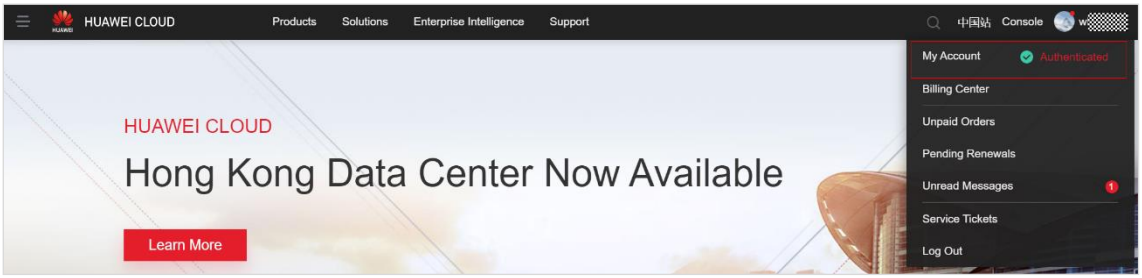


Figure 3-6

Step 2 On the page displayed, click **Manage**.

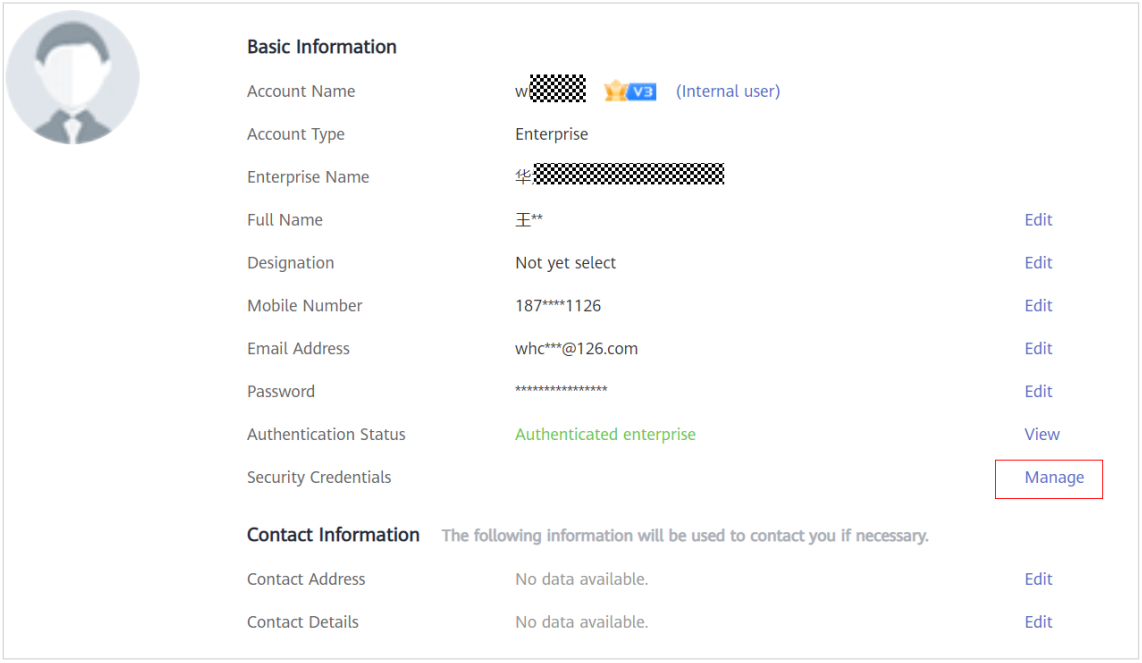


Figure 3-7

Step 3 In the navigation pane, choose **Access Keys**.

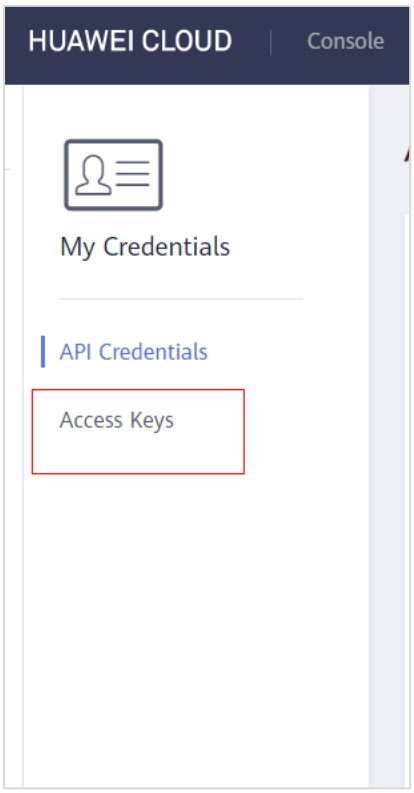


Figure 3-8

Step 4 On the **Access Keys** page, click **Create Access Key** to add an access key. You can also edit, disable, or delete an access key on the right.

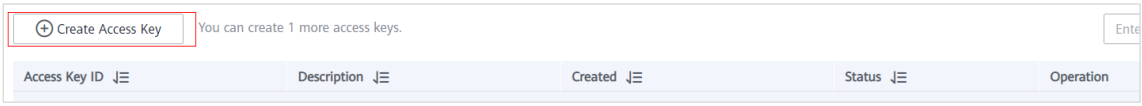


Figure 3-9

Note: After an access key is added, a file containing the access key is automatically downloaded. Keep the file secure.

3.4 Common Huawei Cloud Products

3.4.1 ECS

Elastic Cloud Server (ECS) is a cloud server that provides scalable, on-demand computing resources for secure, flexible, and efficient applications.

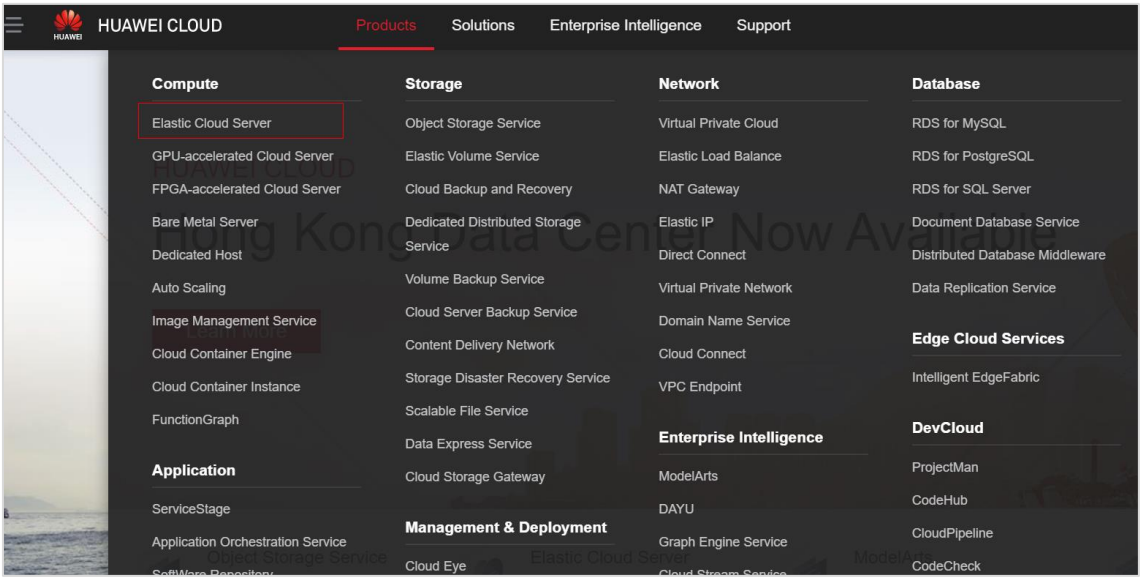


Figure 3-10

On the ECS, you can purchase the required server configuration or even the AI acceleration server (powered by Ascend processors, which will be used in the experiment in chapter 6).

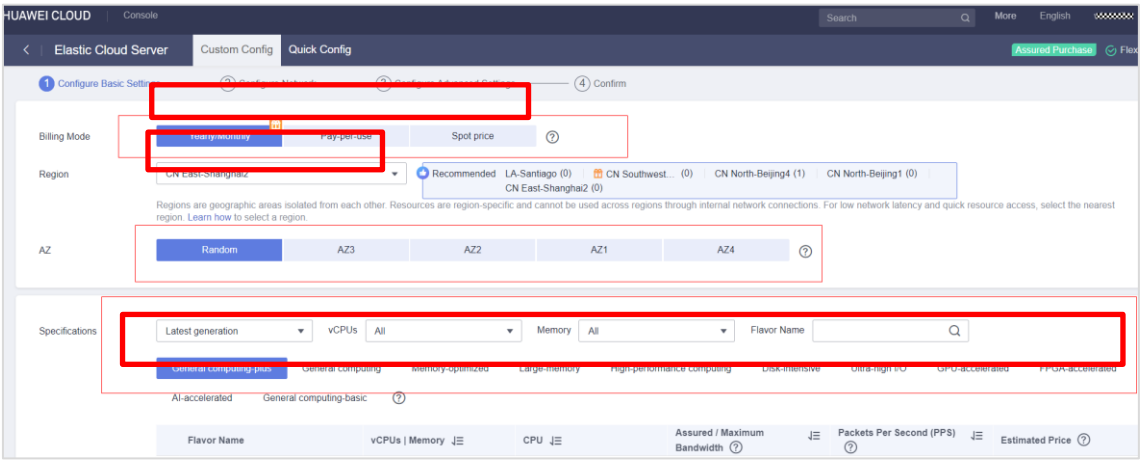


Figure 3-11

3.4.2 ModelArts

ModelArts is a one-stop AI development platform. It provides 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 life cycle during machine learning and deep learning.

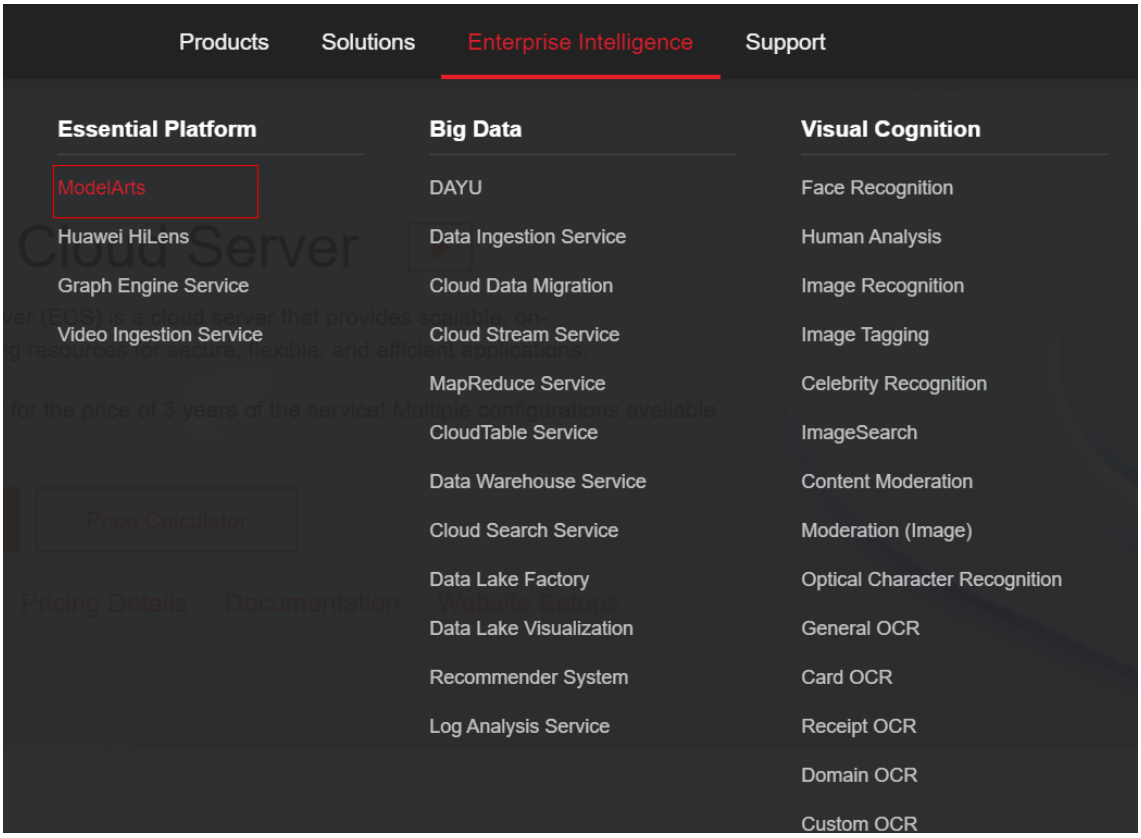


Figure 3-12

After entering ModelArts, you can select multiple GPU-accelerated instances and various services provided by ModelArts. For details, see chapter 8.

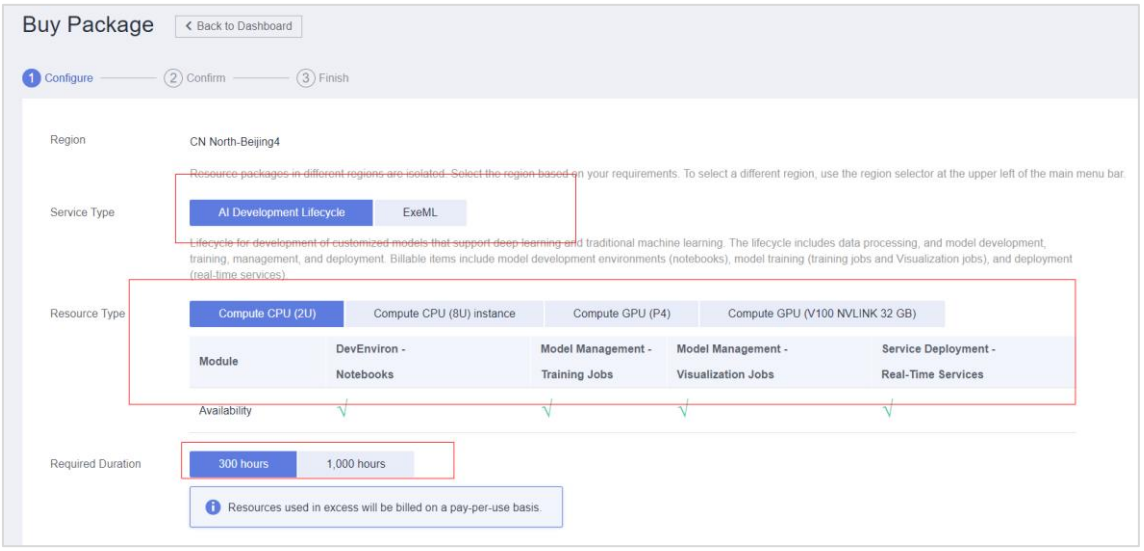


Figure 3-13

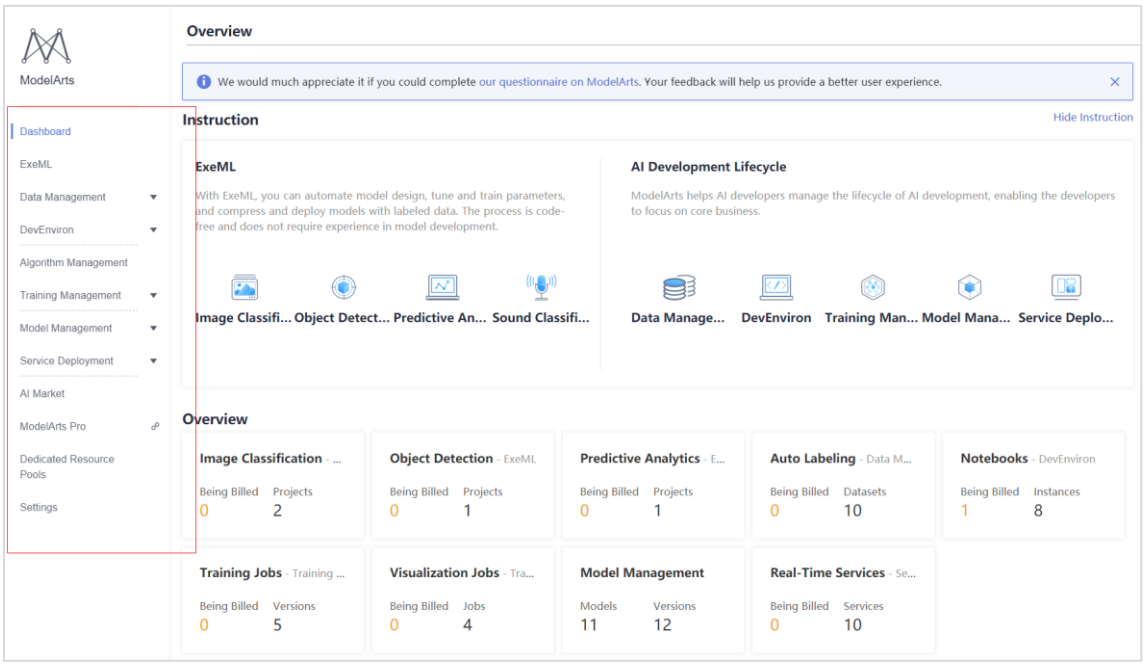


Figure 3-14

Huawei AI Certification Training

HCIA-AI

Machine Learning Experiment Guide

ISSUE:3.0

HUAWEI TECHNOLOGIES CO., LTD.

Huawei Technologies Co., Ltd.

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

Website: http://e.huawei.com

Huawei Certificate System

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

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

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

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


Huawei Certification Portfolio
HUAWEI
Huawei Certification
ICT Vertical Certification
Finance
Public Safety
Platform and Service Certification
Big Data
AI
IoT
Intelligent Vision
Enterprise Communication
GaussDB
Kunpeng Application Developer
Cloud Computing
Cloud Service
ICT Infrastructure Certification
Data Center
Storage
Intelligent Computing
Security
Datacom
WLAN
SDN
Transmission
Access
LTE
5G
HCIE
Huawei Certified ICT Expert
HCIP
Huawei Certified ICT Professional
HCIA
Huawei Certified ICT Associate

About This Document

Overview

This document is applicable to the candidates who are preparing for the HCIA-AI exam and the readers who want to understand the AI programming basics. After learning this guide, you will be able to perform basic 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 Boston House Price Forecast

1.1 Introduction

1.1.1 About This Experiment

The development in this experiment is based on ModelArts. For details about how to set up the environment, see the HCIA-AI V3.0 Experiment Environment Setup Guide. The sample size of the dataset used in this case is small, and the data comes from the open source Boston house price data provided by scikit-learn. The Boston House Price Forecast project is a simple regression model, through which you can learn some basic usage of the machine learning library sklearn and some basic data processing methods.

1.1.2 Objectives

- Upon completion of this task, you will be able to:
  Use the Boston house price data set that is open to the Internet as the model input data.
- Build, train, and evaluate machine learning models.
- Understand the overall process of building a machine learning model.
- Master the application of machine learning model training, grid search, and evaluation indicators.
- Master the application of related APIs.

1.1.3 Datasets and Frameworks Used for the Experiment

This case is based on the Boston dataset, which contains 13 features and 506 data records. Each data record contains detailed information about the house and its surroundings. Specifically, it includes urban crime rate, nitric oxide concentration, average rooms in a house, weighted distance to the downtown area and average house price. The details are as follows:

- CRIM: urban per capita crime rate
- ZN: proportion of residential land exceeds 25,000 square feet
- INDUS: proportion of non-retail commercial land in a town
- CHAS: Charles river empty variable (1 indicates that the boundary is a river; otherwise, the value is 0)
- NOX: Nitric oxide concentration
- RM: average number of rooms in a house

- AGE: proportion of private houses completed before 1940
- DIS: weighted distance to the five central regions of Boston
- RAD: proximity index of a radial highway
- TAX: full value property tax rate of $10,000
- PTRATIO: proportion of teachers and students in urban areas
- target: average price of private houses, unit: $1,000

Framework: Sklearn, which provides Boston house price data, data set segmentation, standardization, and evaluation functions, and integrates various common machine learning algorithms. In addition, XGboost is used, which is an optimized version of GBDT in the integration algorithm.

1.2 Experiment Code

1.2.1 Introducing Dependencies

Code:

```
#Prevent unnecessary warnings.
import warnings
warnings.filterwarnings("ignore")

#Introduce the basic package of data science.
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import scipy.stats as st
import seaborn as sns
##Set attributes to prevent garbled characters in Chinese.
mpl.rcParams['font.sans-serif'] = [u'SimHei']
mpl.rcParams['axes.unicode_minus'] = False

#Introduce machine learning, preprocessing, model selection, and evaluation indicators.
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

#Import the Boston dataset used this time.
from sklearn.datasets import load_boston

#Introduce algorithms.
from sklearn.linear_model import RidgeCV, LassoCV, LinearRegression, ElasticNet
#Compared with SVC, it is the regression form of SVM.
from sklearn.svm import SVR
#Integrate algorithms.
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
```

1.2.2 Loading the Data Set, Viewing Data Attributes, and Visualizing the Data

Step 1 Load the Boston house price data set and display related attributes.

Code:

```
#Load the Boston house price data set.
boston = load_boston()

#x features, and y labels.
x = boston.data
y = boston.target

#Display related attributes.
print('Feature column name')
print(boston.feature_names)
print("Sample data volume: %d, number of features: %d"% x.shape)
print("Target sample data volume: %d"% y.shape[0])
```

Output:

```
Feature column name
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
'B' 'LSTAT']
Sample data volume: 506; number of features: 13
Target sample data volume: 506
```

Step 2 Convert to the dataframe format.

Code:

```
x = pd.DataFrame(boston.data, columns=boston.feature_names)
x.head()
```

Output:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Figure 1-1 First 5 Data Information

Step 3 Visualize label distribution.

Code:

```
sns.distplot(tuple(y), kde=False, fit=st.norm)
```

Output:

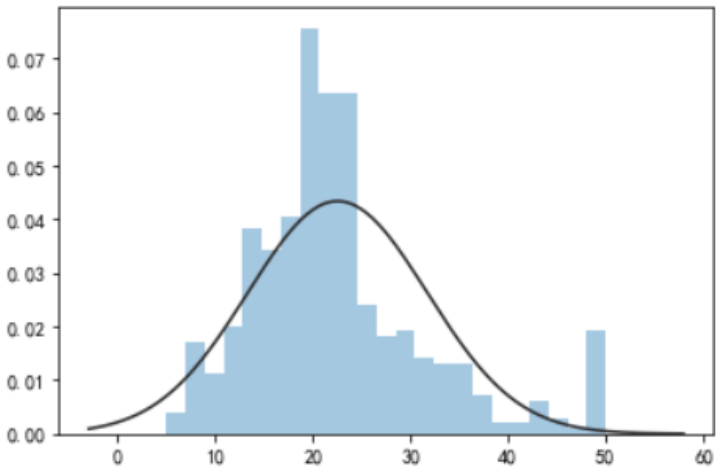


Figure 1-2 Target data distribution

1.2.3 Spliting and Pre-processing the Data Set

Code:

```
#Segment the data.
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=28)
#Standardize the data set.
ss = StandardScaler()
x_train = ss.fit_transform(x_train)
x_test = ss.transform(x_test)
x_train[0:100]
```

Output:

```
array([[-0.35451414, -0.49503678, -0.15692398, ..., -0.01188637,
         0.42050162, -0.29153411],
       [-0.38886418, -0.49503678, -0.02431196, ..., 0.35398749,
         0.37314392, -0.97290358],
       [ 0.50315442, -0.49503678, 1.03804143, ..., 0.81132983,
         0.4391143 , 1.18523567],
       ...,
       [-0.34444751, -0.49503678, -0.15692398, ..., -0.01188637,
         0.4391143 , -1.11086682],
       [-0.39513036, 2.80452783, -0.87827504, ..., 0.35398749,
         0.4391143 , -1.28120919],
       [-0.38081287, 0.41234349, -0.74566303, ..., 0.30825326,
         0.19472652, -0.40978832]])
```

1.2.4 Using Various Regression Models to Model Data Sets

Code:

```
#Set the model name.
names = ['LinerRegression',
        'Ridge',
        'Lasso',
        'Random Forrest',
        'GBDT',
        'Support Vector Regression',
        'ElasticNet',
        'XgBoost']


#Define the model.
# cv is the cross-validation idea here.
models = [LinearRegression(),
          RidgeCV(alphas=(0.001,0.1,1),cv=3),
          LassoCV(alphas=(0.001,0.1,1),cv=5),
          RandomForestRegressor(n_estimators=10),
          GradientBoostingRegressor(n_estimators=30),
          SVR(),
          ElasticNet(alpha=0.001,max_iter=10000),
          XGBRegressor()]
# Output the R2 scores of all regression models.

#Define the R2 scoring function.
def R2(model,x_train, x_test, y_train, y_test):

          model_fitted = model.fit(x_train,y_train)
          y_pred = model_fitted.predict(x_test)
          score = r2_score(y_test, y_pred)
          return score

#Traverse all models to score.
for name,model in zip(names,models):
          score = R2(model,x_train, x_test, y_train, y_test)
          print("{}: {:.6f}, {:.4f}".format(name,score.mean(),score.std()))
```

Output:

```
LinerRegression: 0.564144, 0.0000
Ridge: 0.563700, 0.0000
Lasso: 0.564078, 0.0000
Random Forrest: 0.646657, 0.0000
GBDT: 0.725883, 0.0000
Support Vector Regression: 0.517310, 0.0000
ElasticNet: 0.564021, 0.0000
XgBoost: 0.765266, 0.0000
```

1.2.5 Adjusting Hyperparameters by Grid Search

Step 1 Build a model.

Code:

```
'''
  'kernel': kernel function
   'C': SVR regularization factor
   'gamma': 'rbf', 'poly' and 'sigmoid' kernel function coefficient, which affects the model performance
'''
parameters = {
   'kernel': ['linear', 'rbf'],
    'C': [0.1, 0.5,0.9,1,5],
    'gamma': [0.001,0.01,0.1,1]
}

#Use grid search and perform cross validation.
model = GridSearchCV(SVR(), param_grid=parameters, cv=3)
model.fit(x_train, y_train)
```

Output:

```
GridSearchCV(cv=3, error_score='raise',
       estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'kernel': ['linear', 'rbf'], 'C': [0.1, 0.5, 0.9, 1, 5], 'gamma': [0.001, 0.01, 0.1, 1]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

Step 2 Obtain optimal parameters.

Code:

```
print("Optimal parameter list:", model.best_params_)
print("Optimal model:", model.best_estimator_)
print("Optimal R2 value:", model.best_score_)
```

Output:

```
Optimal parameter list: {'C': 5, 'gamma': 0.1, 'kernel': 'rbf'}
Optimal model: SVR(C=5, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.1,
kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
Optimal R2 value: 0.797481706635164
```

Step 3 Perform visualization.

Code:

```
##Perform visualization.
ln_x_test = range(len(x_test))
y_predict = model.predict(x_test)

#Set the canvas.
plt.figure(figsize=(16,8), facecolor='w')
#Draw with a red solid line.
plt.plot (ln_x_test, y_test, 'r-', lw=2, label=u'Value')
#Draw with a green solid line.
```

```
plt.plot (ln_x_test, y_predict, 'g-', lw = 3, label=u'Estimated value of the SVR algorithm, $R^2$=%.3f' %
(model.best_score_))

#Display in a diagram.
plt.legend(loc ='upper left')
plt.grid(True)
plt.title(u"Boston Housing Price Forecast (SVM)")
plt.xlim(0, 101)
plt.show()
```

Output:

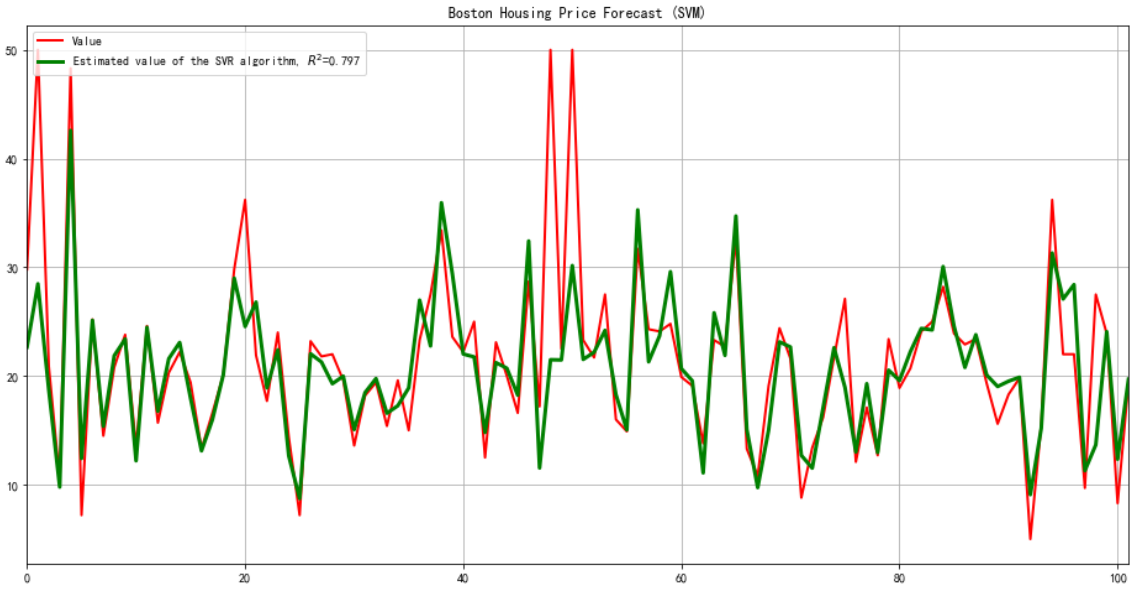


Figure 1-3 Visualization result

1.3 Summary

This chapter describes how to build a Boston house price regression model based on sklearn, including importing, segmenting, and standardizing data, defining models, and setting hyperparameters, and provides trainees with a basic concept of machine learning model building.

2 Detail of linear regression

2.1 Introduction

2.1.1 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.

2.1.2 Objectives

The main purpose of this experiment is as follows.

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

2.2 Experiment Code

2.2.1 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:

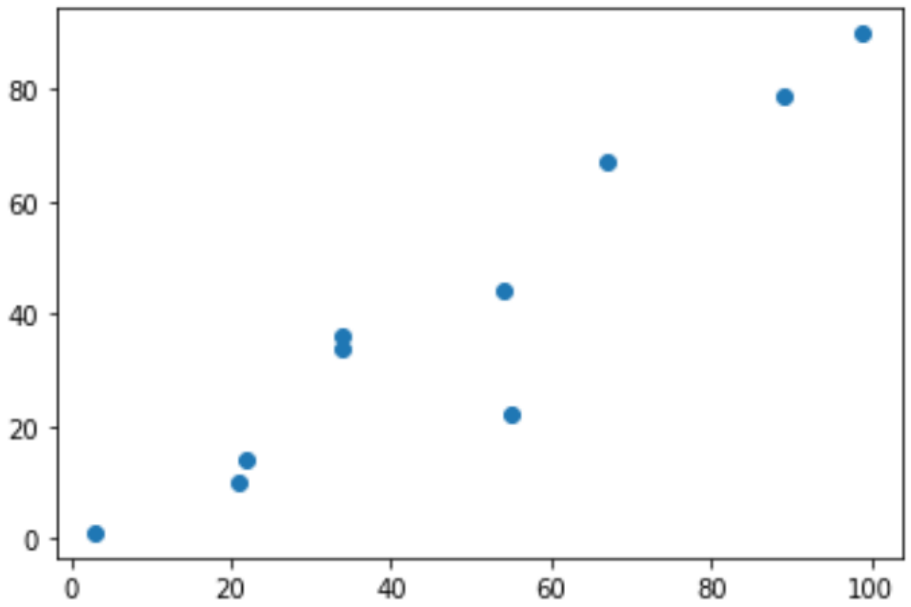


Figure 2-1 Scatter Plot

2.2.2 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
```

2.2.3 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:

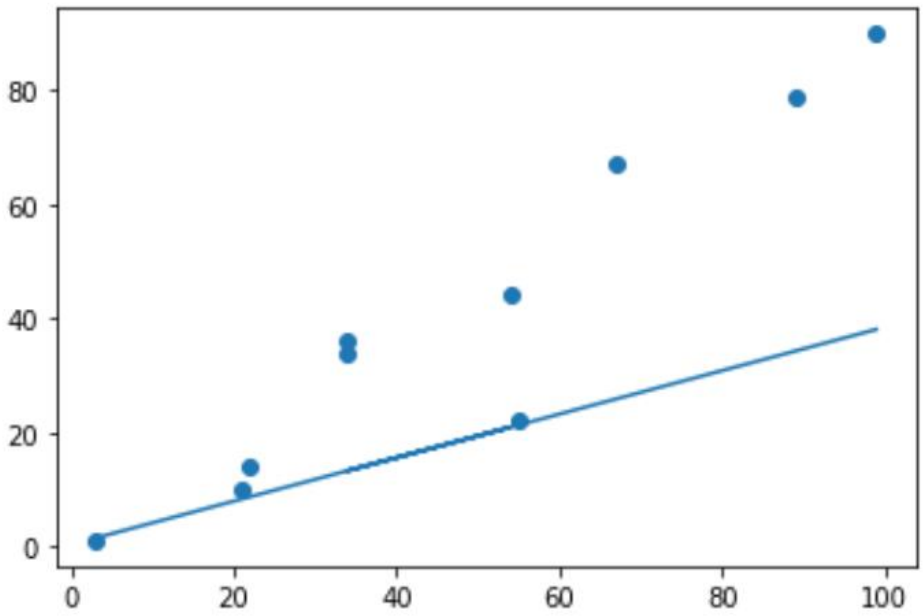


Figure 2-2 Iterate 1 time

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:

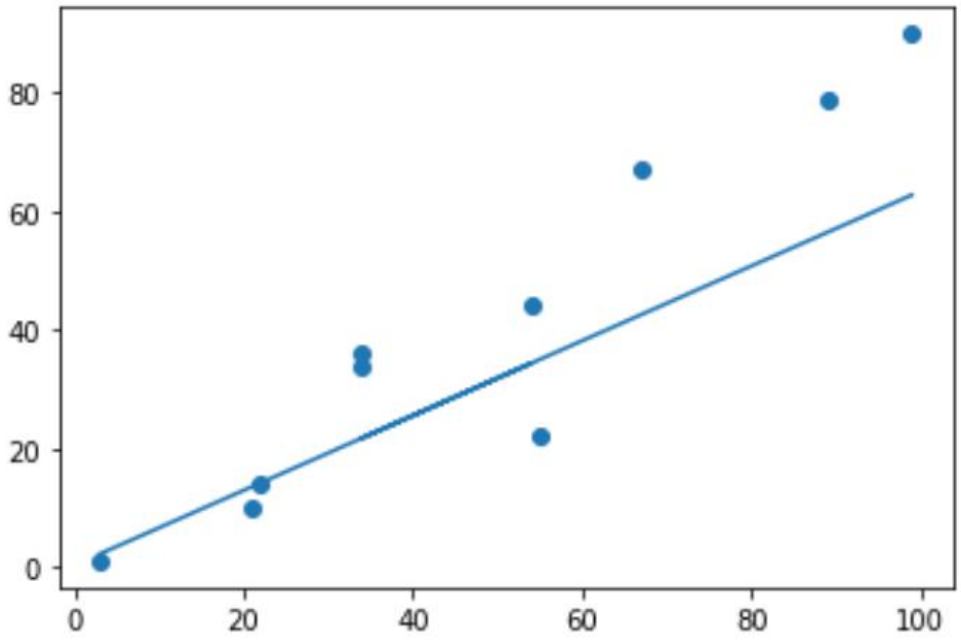


Figure 2-3 Iterate 2 times

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:

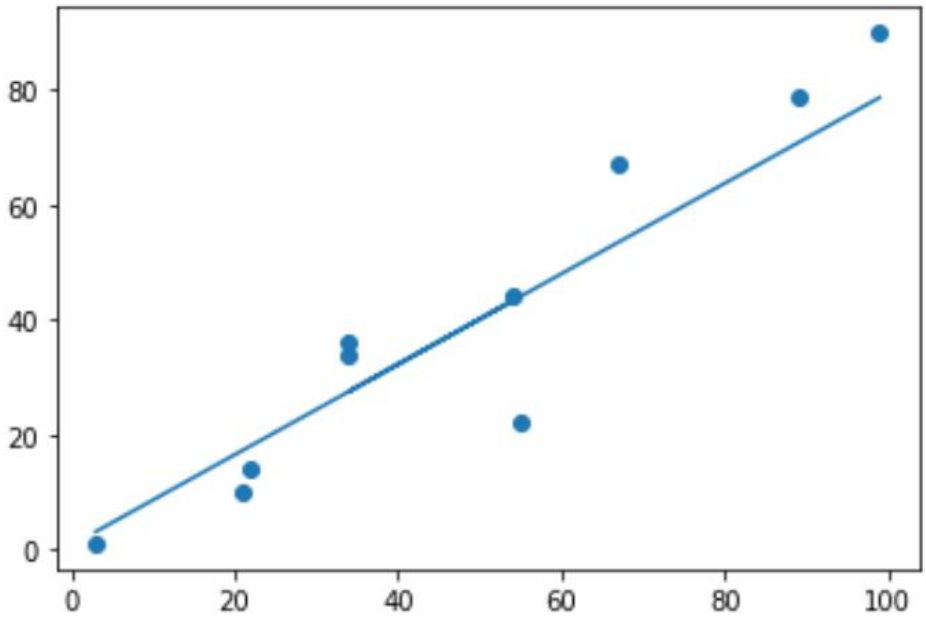


Figure 2-4 Iterate 3 times

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:

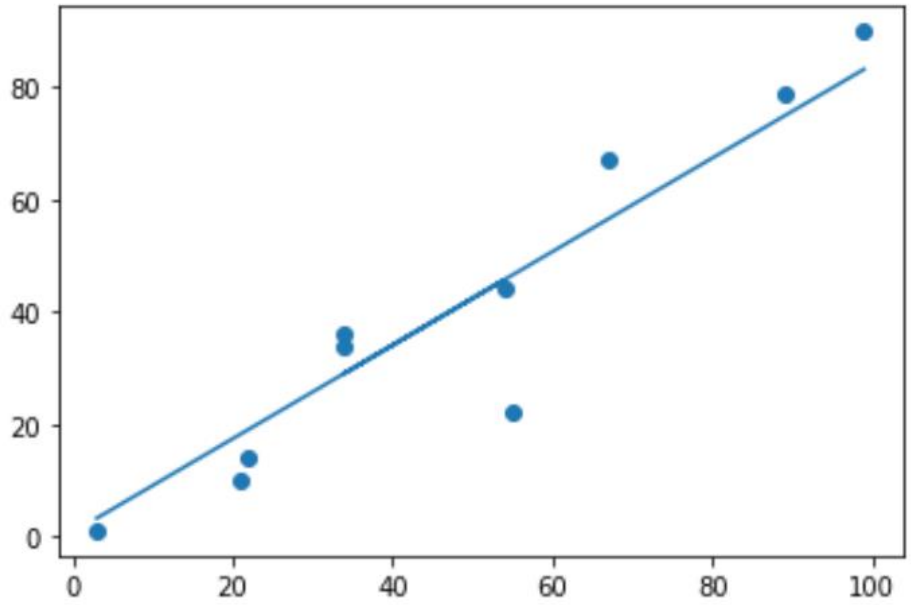


Figure 2-5 Iterate 4 times

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:

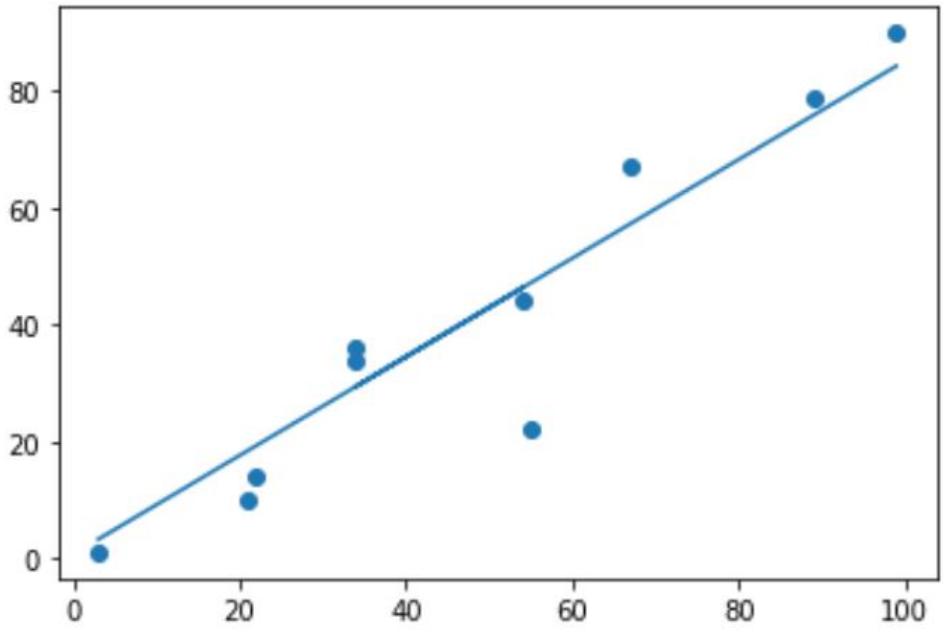


Figure 2-6 Iterate 5 times

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:

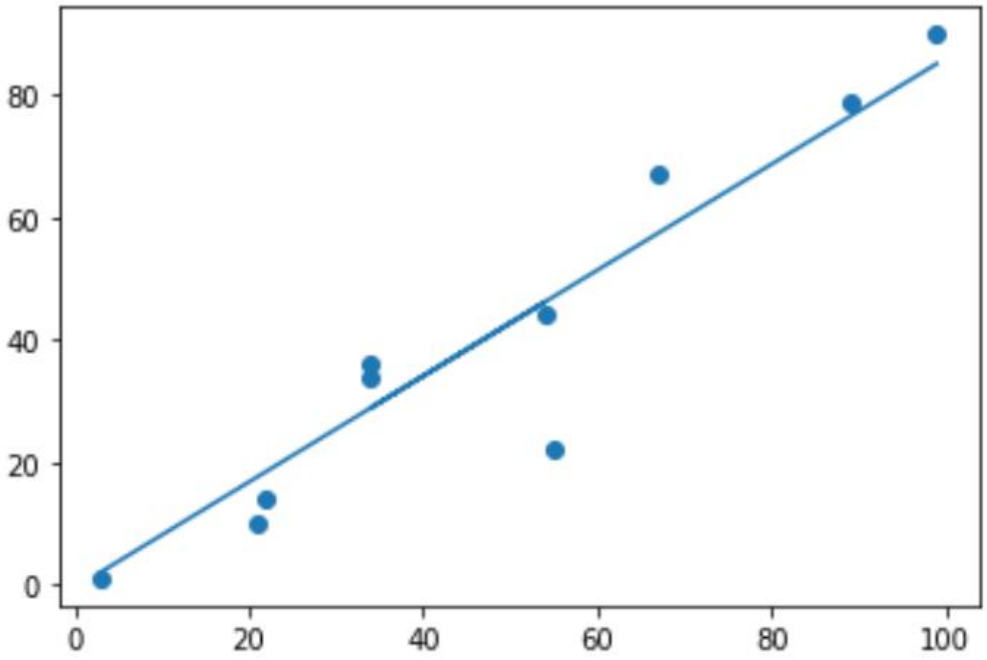


Figure 2-7 Iterate 10000 times

2.3 Thinking and practice

2.3.1 Question 1

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

2.3.2 Question 2

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

3 Decision tree details

3.1 Introduction

3.1.1 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

3.1.2 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

3.2 Experiment Code

3.2.1 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
```

3.2.2 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)
```

3.2.3 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
```

3.2.4 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 3-1 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 3-2 CART tree result

Huawei AI Certification Training

HCIA-AI

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

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

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

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


Huawei Certification Portfolio
HUAWEI
Huawei Certification
ICT Vertical Certification
Finance
Public Safety
Platform and Service Certification
Big Data
AI
IoT
Intelligent Vision
Enterprise Communication
GaussDB
Kunpeng Application Developer
Cloud Computing
Cloud Service
ICT Infrastructure Certification
Data Center
Storage
Intelligent Computing
Security
Datacom
WLAN
SDN
Transmission
Access
LTE
5G
HCIE
Huawei Certified ICT Expert
HCIP
Huawei Certified ICT Professional
HCIA
Huawei Certified ICT Associate

About This Document

Overview

This document is applicable to the candidates who are preparing for the HCIA-AI exam or readers who want to understand AI basics, TensorFlow basics and AI programming basics. After learning this guide, you will be able to perform basic AI image recognition programming.

Description

This experiment guide introduces the following three experiments:

- Experiment 1: TensorFlow basics
  - This experiment mainly describes the basic syntax of TensorFlow 2.
- Experiment 2: common modules of TensorFlow 2
  - This experiment mainly introduces Keras interfaces.
- Experiment 3: handwritten text recognition
  - This experiment uses basic code to help learners understand how to implement handwritten text recognition through TensorFlow 2.0.
- Experiment 4: Image Classification
  - This experiment is based on how to use TensorFlow 2 and python packages to predict image categories from CIFAR10 image classification dataset. It is hoped that trainees or readers can get started with deep learning and have the basic programming capability of implementing image recognition models.

Background Knowledge Required

The readers must be able to:

- Know basic Python knowledge.
- Understand basic TensorFlow concepts.
- Command basic Python programming.
- knowledge of data structures and deep learning algorithms.

Experiment Environment Overview

Python Development Tool

This experiment environment is developed and compiled based on Python 3.6 and TensorFlow 2.1.0.

Contents

1 TensorFlow 2.x Basics

1.1 Introduction

1.1.1 About This Experiment

This experiment helps trainees understand the basic syntax of TensorFlow 2.x by introducing a series of tensor operations of TensorFlow 2.x, including tensor creation, slicing, and indexing, tensor dimension modification, tensor arithmetic operations, and tensor sorting.

1.1.2 Objectives

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

- Understand how to create a tensor.
- Master the tensor slicing and indexing methods.
- Master the syntax for tensor dimension modification.
- Master arithmetic operations of tensors.
- Master the tensor sorting method.
- Dive deeper into eager execution and AutoGraph based on code.

1.2 Experiment Steps

1.2.1 Introduction to Tensors

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

- A defined constant tensor has an unchangeable value and dimension, and a defined variable tensor has a changeable value and an unchangeable dimension.
- In neural networks, variable tensors are generally used as matrices for storing weights and other information, and are a type of trainable data. Constant tensors can be used for storing hyperparameters or other structured data.

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(), and tf.ones_like(): create 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.
- Creating a list object by using NumPy, and then converting the list object into a tensor by using tf.convert_to_tensor.

Step 1 tf.constant()

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

- value: A constant value (or list) of output type dtype.
- dtype: The type of the elements of the resulting tensor.
- shape: Optional dimensions of resulting tensor.
- name: Optional name for the tensor.

Code:

```
import tensorflow as tf
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:

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

Code:

```
#View common attributes.
print("value of the constant const_a:", const_a.numpy())
print("data type of the constant const_a:", const_a.dtype)
print("shape of the constant const_a:", const_a.shape)
print("name of the device that is to generate the constant const_a:", const_a.device)
```

Output:

```
Value of the constant const_a: [[1. 2.]
 [3. 4.]]
Data type of the constant const_a: <dtype: 'float32'>
Shape of the constant const_a: (2, 2)
Name of the device that is to generate the constant const_a:
/job:localhost/replica:0/task:0/device:CPU:0
```

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

Usages of **tf.ones()** and **tf.ones_like()** are similar to those of **tf.zeros()** and **tf.zeros_like()**. Therefore, the following describes only the usages of **tf.ones()** and **tf.ones_like()**.

Create a constant with the value **0**.

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

- shape: A list of integers, a tuple of integers, or a 1-D Tensor of type int32.t
- dtype: The DType of an element in the resulting Tensor.
- name: Optional string. A name for the operation.

Code:

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

Create a tensor whose value is **0** based on the input tensor, with its shape being the same as that of the input tensor.

tf.zeros_like(input, dtype=None, name=None):

- input_tensor: A Tensor or array-like object.
- dtype: A type for the returned Tensor. Must be float16, float32, float64, int8, uint8, int16, uint16, int32, int64, complex64, complex128, bool or string (optional).
- name: A name for the operation (optional).

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 scalar value.

tf.fill(dims, value, name=None):

- dims: A 1-D sequence of non-negative numbers. Represents the shape of the output tf.Tensor. Entries should be of type: int32, int64.
- value: A value to fill the returned tf.Tensor.
- name: Optional string. The name of the output tf.Tensor.

Code:

```
fill_d = tf.fill([3,3], 8) # Create a 2x3 matrix with all values being 8.
#View data.
fill_d.numpy()
```

Output

```
array([[8, 8, 8],
       [8, 8, 8],
       [8, 8, 8]])
```

Step 4 tf.random

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

Create a tensor that conforms to a normal distribution.

tf.random.normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32,seed=None, name=None):

- shape: A 1-D integer Tensor or Python array. The shape of the output tensor.
- mean: A Tensor or Python value of type dtype, broadcastable with stddev. The mean of the normal distribution.
- stddev: A Tensor or Python value of type dtype, broadcastable with mean. The standard deviation of the normal distribution.
- dtype: The type of the output.
- seed: A Python integer. Used to create a random seed for the distribution. See tf.random.set_seed for behavior.
- name: A name for the operation (optional).

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 the list object into a tensor by using **tf.convert_to_tensor**.

This method can convert a given value into a tensor. **tf.convert_to_tensor** can be used to convert a Python data type into a tensor data type available to TensorFlow.

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

- value: An object whose type has a registered Tensor conversion function.
- dtype: Optional element type for the returned tensor. If missing, the type is inferred from the type of value.
- 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. If the conversion to dtype_hint is not possible, this argument has no effect.
- Name: Optional name to use if a new Tensor is created.

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 operated using the **tf.Variable** class. **tf.Variable** indicates a tensor. The value of **tf.Variable** can be changed by running an arithmetic operation on **tf.Variable**. Variable values can be read and changed.

Code:

```
#Create a variable. Only the initial value needs to be provided.
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 the variable var_1:",var_1.read_value())
#Assign a variable value.
var_value_1=[[1,2,3],[4,5,6]]
var_1.assign(var_value_1)
print("Value of the variable var_1 after the assignment:",var_1.read_value())
```

Output:

```
Value of the variable var_1: tf.Tensor(
[[1. 1. 1.]
 [1. 1. 1.]], shape=(2, 3), dtype=float32)
Value of the variable var_1 after the assignment: tf.Tensor(
[[1. 2. 3.]
 [4. 5. 6.]], shape=(2, 3), dtype=float32)
```

Code:

```
#Variable addition
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

Tensor slicing methods include:

- [start: end]: extracts a data slice from the start position to the end position of the tensor.
- [start:end:step] or [::step]: extracts a data slice at an interval of step from the start position to the end position of the 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 at an interval of 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 position [20,40] 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 of data 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, batch_dims=0, name=None):

- params: A Tensor. The tensor from which to gather values.
- indices: A Tensor. Must be one of the following types: int32, int64. Index tensor.
- Name: A name for the operation (optional).
- batch_dims: An integer or a scalar 'Tensor'. The number of batch dimensions.

Code:

```
#Extract the pixel in [1,1] from the first dimension of the first image and the pixel in [2,2] from 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, and **tf.shape(x)** returns Tensor objects.

1.2.1.3.2 Dimension Reshaping

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

- tensor: input tensor
- shape: dimension 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. When the number of dimensions of the input data is D, the axis must fall in the range of [–(D + 1), D] (included). A negative value indicates adding a dimension in 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("size of the original data:",expand_sample_1.shape)
print("add a dimension before the first dimension (axis = 0): ",tf.expand_dims(expand_sample_1, axis=0).shape)
print("add a dimension before the second dimension (axis = 1): ",tf.expand_dims(expand_sample_1, axis=1).shape)
print("add a dimension after the last dimension (axis = –1): ",tf.expand_dims(expand_sample_1, axis=-1).shape)
```

Output:

```
Size of the original data: (100, 100, 3)
Add a dimension before the first dimension (axis = 0): (1, 100, 100, 3)
Add a dimension before the second dimension (axis = 1): (100, 1, 100, 3)
Add a dimension after the last dimension (axis = –1): (100, 100, 3, 1)
```

1.2.1.3.4 Dimension Squeezing

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

- input: input tensor
- axis: If axis is set to 1, dimension 1 needs to be deleted.

Code:

```
#Generate a 100 x 100 x 3 tensor to represent a 100 x 100 three-channel color image.
squeeze_sample_1 = tf.random.normal([1,100,100,3])
print("size of the original data:",squeeze_sample_1.shape)
squeezed_sample_1 = tf.squeeze(expand_sample_1)
print("data size after dimension squeezing:",squeezed_sample_1.shape)
```

Output:

```
Size of the original data: (1, 100, 100, 3)
```

```
Data size after dimension squeezing: (100, 100, 3)
```

1.2.1.3.5 Transpose

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

- a: input tensor
- perm: tensor size sequence, generally used to transpose high-dimensional arrays
- conjugate: indicates complex number transpose.
- name: tensor name

Code:

```
#Input the tensor to be transposed, and call tf.transpose.
trans_sample_1 = tf.constant([1,2,3,4,5,6],shape=[2,3])
print("size of the original data:",trans_sample_1.shape)
transposed_sample_1 = tf.transpose(trans_sample_1)
print("size of transposed data:",transposed_sample_1.shape)
```

Output:

```
Size of the original data: (2, 3)
Size of the transposed data: (3, 2)
```

perm is required for high-dimensional data transpose, and indicates the dimension sequence of the input tensor.

The original dimension sequence of a three-dimensional tensor is [0, 1, 2] (**perm**), indicating the length, width, and height of high-dimensional data, respectively.

Data dimensions can be transposed by changing the sequence of values in **perm**.

Code:

```
#Generate an $ 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("size of the original data:",trans_sample_2.shape)
#Exchange the length and width for the four images: The original perm value is [0,1,2,3], and the new perm value is [0,2,1,3].
transposed_sample_2 = tf.transpose(trans_sample_2,[0,2,1,3])
print("size of transposed data:",transposed_sample_2.shape)
```

Output:

```
Size of the original data: (4, 100, 200, 3)
Size of the transposed data: (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("broadcasted data:",broadcasted_sample_1.numpy())
```

Output:

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

Main 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 describes only one addition example.

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 tensors.
- tf.nn.in_top_k(prediction, target, K): calculates whether the predicted value is equal to the actual value, and returns a Boolean tensor.

The following describes how to use **tf.argmax()**:

Return the position of the maximum value.

tf.argmax(input,axis):

- input: input tensor
- axis: maximum output value in 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, a series of operations of **tf.reduce_*** reduce tensor dimensions. The series of operations can be performed on dimensional 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 value), **tf.reduce_all** (logical AND), **tf.reduce_any** (logical OR), and **tf.reduce_logsumexp** (**log(sum(exp))**).

The methods for using these operations are similar. The following describes how to use **tf.reduce_sum**.

Calculate the sum of elements in all dimensions of a tensor.

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

- input_tensor: The tensor to reduce. Should have numeric type.
- axis: The dimensions to reduce. If None (the default), reduces all dimensions. Must be in the range [-rank(input_tensor),rank(input_tensor)].
- keepdims: If true, retains reduced dimensions with length 1.
- name: A name for the operation (optional).

Code:

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

Output:

```
Original data [[1 2 3]
 [4 5 6]]
Calculate the sum of all elements in the tensor (axis = None): 21
Calculate the sum of elements in each column by row (axis = 0): [5 7 9]
Calculate the sum of elements in each column by 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 vectors based on the specified dimension, while keeping other dimensions unchanged.
- tf.stack(): changes a group of R dimensional tensors to R+1 dimensional tensors, with the dimensions changed after the concatenation.
- tf.concat(values, axis, name='concat'):
- values: input tensor
- axis: dimension to concatenate
- name: operation name

Code:

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

Output:

```
Sizes of the original data: (4, 100, 100, 3) (40, 100, 100, 3)
Size of the concatenated data: (44, 100, 100, 3)
```

A dimension can be added to an original matrix in the same way. **axis** determines the position of the dimension.

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

- values: A list of Tensor objects with the same shape and type.
- axis: An int. The axis to stack along. Defaults to the first dimension. Negative values wrap around, so the valid range is [-(R+1), R+1).
- name: A name for this operation (optional).

Code:

```
stack_sample_1 = tf.random.normal([100,100,3])
stack_sample_2 = tf.random.normal([100,100,3])
print("sizes of the original data: ",stack_sample_1.shape, stack_sample_2.shape)
#Dimensions increase after the concatenation. 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("size of the concatenated data:",stacked_sample_1.shape)
```

Output:

```
Sizes of the original data: (100, 100, 3) (100, 100, 3)
Size of the concatenated data: (2, 100, 100, 3)
```

1.2.1.6.2 Tensor Splitting

In TensorFlow, tensor splitting operations include:

- tf.unstack(): splits a tensor by a specific dimension.
- tf.split(): splits a tensor into a specified number of sub tensors based on a specific dimension.
- tf.split() is more flexible than tf.unstack().
- tf.unstack(value,num=None,axis=0,name='unstack'):
- value: input tensor
- num: indicates that a list containing num elements is output. The value of num must be the same as the number of elements in the specified dimension. This parameter can generally be ignored.
- axis: specifies the dimension based on which the tensor is split.
- name: operation name

Code:

```
#Split data based on the first dimension and output the split 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, num=None, name='split'):

- value: The Tensor to split.
- num_or_size_splits: Either an integer indicating the number of splits along axis or a 1-D integer Tensor or Python list containing the sizes of each output tensor along axis. If a scalar, then it must evenly divide value.shape[axis]; otherwise the sum of sizes along the split axis must match that of the value.
- axis: An integer or scalar int32 Tensor. The dimension along which to split. Must be in the range [-rank(value), rank(value)). Defaults to 0.
- num: Optional, used to specify the number of outputs when it cannot be inferred from the shape of size_splits.
- name: A name for the operation (optional).

tf.split() splits a tensor in either of the following ways:

- If the value of num_or_size_splits is an integer, the tensor is evenly split into sub tensors in the specified dimension (axis = D).
- If the value of num_or_size_splits is a vector, the tensor is split into sub tensors based on the element value of the vector in the specified dimension (axis = D).

Code:

```
import numpy as np
split_sample_1 = tf.random.normal([10,100,100,3])
print("size of the original data:",split_sample_1.shape)
splited_sample_1 = tf.split(split_sample_1, num_or_size_splits=5,axis=0)
print("size of the split data when m_or_size_splits is set to 10: ",np.shape(splited_sample_1))
splited_sample_2 = tf.split(split_sample_1, num_or_size_splits=[3,5,2],axis=0)
print("sizes of the split data when num_or_size_splits is set to [3,5,2]:",
      np.shape(splited_sample_2[0]),
      np.shape(splited_sample_2[1]),
      np.shape(splited_sample_2[2]))
```

Output:

```
Size of the original data: (10, 100, 100, 3)
Size of the split data when m_or_size_splits is set to 10: (5, 2, 100, 100, 3)
Sizes of the split data when num_or_size_splits is set to [3,5,2]: (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 tensor indexes.
- tf.nn.top_k(): returns the first k maximum values.
- tf.sort/argsort(input, direction, axis):
- input: input tensor
- direction: sorting order, which can be set to DESCENDING (descending order) or ASCENDING (ascending order). The default value is ASCENDING.
- axis: sorting by the dimension specified by axis. The default value of axis is –1, indicating the last dimension.

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("indexes 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]
Indexes of elements in ascending order: [9 0 7 8 6 5 4 2 1 3]
```

tf.nn.top_k(input,k=1,sorted=True,name=None):

- input: 1-D or higher Tensor with last dimension at least k.
- K: 0-D int32 Tensor. Number of top elements to look for along the last dimension (along each row for matrices).
- sorted: If true the resulting k elements will be sorted by the values in descending order.
- name: Optional name for the operation.

Return two tensors:

- values: k maximum values in each row
- indices: positions of elements in the last dimension of the input tensor

Code:

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

Output:

```
Input tensor: [1 8 7 9 6 5 4 2 3 0]
First five values in ascending order: [9 8 7 6 5]
```

```
Indexes of the first five values in ascending order: [3 1 2 4 5]
```

1.2.2 Eager Execution of TensorFlow 2.x

Eager execution mode:

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

Graph mode:

TensorFlow 1.0 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 2.0 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)
#Enable the drawing function, 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 2.0 and enable the eager execution mode. Another advantage of the eager execution mode lies in availability of native Python functions, for example, 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 graph mode.

1.2.3 AutoGraph of TensorFlow 2.x

When used to comment out a function, the decorator **tf.function** 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 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 viewed directly, but the return value can be viewed using **.numpy()**.

The following compares the performance of the graph mode and eager execution mode by performing the same operation (computation of one CNN 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 running the operation at one CNN layer in eager execution mode:
18.26327505100926
Time required for running the operation at 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 function can be used to improve the code execution efficiency.

2 Common Modules of TensorFlow 2.x

2.1 Introduction

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

- tf.data: implements operations on datasets.
- These operations include reading datasets directly from the memory, reading CSV files, reading TFRecord files, and augmenting data.
- tf.image: implements processing operations on images.
- 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 task, you will be able to master the common deep learning modeling interfaces in **tf.keras**.

2.3 Experiment Steps

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.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'))
```

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 such as:

- 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 previous 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**.

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

- activation: sets the activation function for the layer. By default, the system applies no activation function.
- kernel_initializer and bias_initializer: initialization schemes that create the layer's weights (kernel and bias). This defaults to the Glorot uniform initializer.
- kernel_regularizer and bias_regularizer: regularization schemes that apply to the layer's weights (kernel and bias), such as L1 or L2 regularization. By default, the system applies no regularization function.

2.3.1.3.1 tf.keras.layers.Dense

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

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

Code:

```
#Create a fully connected layer that contains 32 neurons. 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 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 step
- 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: sets the activation function.
- data_format: data format, set to 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 (128, 128, 3) if the value is channels_last. The default value of this parameter is the value specified in ~/.keras/keras.json. If this parameter 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.MaxPool2D/AveragePool2D

Main configuration parameters in **tf.keras.layers.MaxPool2D**/**AveragePool2D** include, :

- pool_size: size of the pooled kernel. For example, if the matrix (2, 2) is used, the picture becomes half of the original length in both dimensions. If this parameter is set to an integer, the integer is the values of all dimensions.
- strides: Integer, tuple of 2 integers, or None. Strides values. Specifies how far the pooling window moves for each pooling step. If None, it will default to pool_size.
- padding: One of "valid" or "same" (case-insensitive). "valid" adds no zero padding. "same" adds padding such that if the stride is 1, the output shape is the same as input shape.

- data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

Code:

```
layers.MaxPoo2D(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
- activation: sets the activation function.
- recurrent_activation: activation function to use for the recurrent step
- return_sequences: If the value is True, the system returns the full sequence. If the value is False, the system returns the output in the last cell of the output sequence.
- 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 is built, you can call **compile** to configure the learning process of the model:

compile(optimizer='rmsprop', loss=None, metrics=None, loss_weights=None,

weighted_metrics=None, run_eagerly=None, **kwargs):

- optimizer: String (name of optimizer) or optimizer instance.
- 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, set 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.
- weighted_metrics: List of metrics to be evaluated and weighted by sample_weight or class_weight during training and testing.
- run_eagerly: Bool. Defaults to False. If True, this Model's logic will not be wrapped in a tf.function. Recommended to leave this as None unless your Model cannot be run inside a tf.function.
- **kwargs: Any additional arguments. Supported arguments:

  experimental_steps_per_execution: Int. The number of batches to run during each tf.function call. Running multiple batches inside a single tf.function call can greatly improve performance on TPUs or small models with a large Python overhead. Note that if this value is set to N, Callback.on_batch methods will only be called every N batches. This currently defaults to 1. At most, one full epoch will be run each execution. If a number larger than the size of the epoch is passed, the execution will be truncated to the size of the epoch.

  sample_weight_mode for backward compatibility.

Code:

```
model = tf.keras.Sequential()
model.add(layers.Dense(10, activation='softmax'))
#Determine the optimizer (optimizer), loss function (loss), 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, validation_batch_size=None, validation_freq=1, max_queue_size=10, workers=1, use_multiprocessing=False):

- x: input training data
- y: target (labeled) data
- batch_size: number of samples for each gradient update The default value is 32.
- epochs: number of iteration rounds of the training model
- verbose: log display mode, set to 0, 1, or 2.
  - 0: no display
  - 1: progress bar
  - 2: one line for each round
- callbacks: callback function used during training
- validation_split: fraction of the training data to be used as validation data
- validation_data: validation set. This parameter will overwrite validation_split.
- shuffle: indicates 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 validate before stopping. This parameter is valid only when steps_per_epoch is specified.
- validation_batch_size: Integer or None. Number of samples per validation batch
- validation_freq:Only relevant if validation data is provided. Integer or collections_abc.
- max_queue_size Integer. Used for generator or keras.utils.Sequence input only. Maximum size for the generator queue. If unspecified, max_queue_size will default to 10.
- workers: Integer. Used for generator or keras.utils.Sequence input only. Maximum number of processes to spin up when using process-based threading. If unspecified, workers will default to 1. If 0, will execute the generator on the main thread.
- use_multiprocessing: Boolean. Used for generator or keras.utils.Sequence input only. If True, use process-based threading. If unspecified, use_multiprocessing will

default to False. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

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>
```

You can use **tf.data** to build training input pipelines for large datasets.

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: exports and visualizes the training progress and results with TensorBoard.

Code:

```
#Set hyperparameters.
Epochs = 10

#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',
        #Whether to save the optimal model.
        save_best_only=True,
        #Monitored metric
        monitor='val_loss'),

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

```
    #Use TensorBoard.
    tf.keras.callbacks.TensorBoard(log_dir='./logs')
]

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
import os
# create the file
if not os.path.exists('./model/'):
    os.mkdir('./model/')
#Save models.
model.save('./model/the_save_model.h5')
#Import models.
new_model = tf.keras.models.load_model('./model/the_save_model.h5')
new_prediction = new_model.predict(test_x)
#np.testing.assert_allclose: determines whether the similarity between 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, or 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 text in handwriting images. Different from printed fonts, handwriting of different people has different sizes and styles, making it difficult for computers to recognize handwriting.

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

3.2 Objectives

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

- Master the basic process of TensorFlow computing.
- Be familiar with the basic elements of network building, including dataset, network model building, model training, and model validation.

3.3 Experiment Steps

This experiment involves the following steps:

- Reading the MNIST handwritten digit dataset.
- Getting started with TensorFlow by using simple mathematical models.
- Implementing softmax regression by using high-level APIs.
- Building a multi-layer CNN.
- Implementing a CNN by using high-level APIs.
- Visualizing prediction results.

3.3.1 Project Description and Dataset Acquisition

3.3.1.1 Description

This project applies deep learning and TensorFlow tools to train and build models based on the MNIST handwriting dataset.

3.3.1.2 Data Acquisition and Processing

3.3.1.2.1 About the Dataset

The MNIST dataset is provided by the National Institute of Standards and Technology (NIST).

The dataset consists of handwritten digits from 250 individuals, of which 50% are high school students and 50% are staff from Bureau of the Census.

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)

The MNIST dataset is an entry-level computer vision dataset that contains images of various handwritten digits.

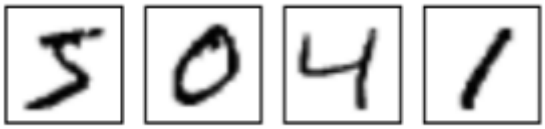


Figure 3-1 The MNIST Dataset

It also contains one label for 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 directly 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 are 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**.

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 may 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 9 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:

Figure 3-2 First 9 Images

3.3.2.2 Data Preprocessing

An 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 into 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:

```
#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 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
```

The evaluation shows that the model accuracy reaches 0.87, and 10 training iterations have been performed.

3.3.3.5 Saving the DNN Model

Create **model** folder under relative path.

Code:

```
model.save('./model/final_DNN_model.h5')
```

3.3.4 CNN Construction

The conventional CNN construction method helps you better understand the internal network structure but has 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 two pooling layers are first added by using **keras.layers**. Afterwards, a dropout layer is added to prevent overfitting. Finally, two fully connected 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 verification shows that accuracy of the CNN model reaches up to 99%.

3.3.4.4 Saving the CNN Model

Code:

```
model.save('./model/final_CNN_model.h5')
```

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('./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 prediction results.

Code:

```
#Visualize test set output results.
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 set 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')
        #Visualize image display.
```

```
            img = X_test[i].reshape((28,28))#Read each row of data in the format of Ndarry.
            plt.axis("off")
            ax[i].imshow(img, cmap='Greys', interpolation='nearest')#Visualization
            ax[i].axis("off")
        print('first {} images in the test set:'.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,
```

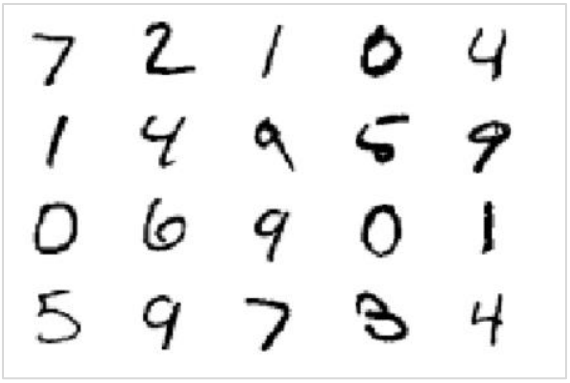


Figure 3-3 First 20 images of the test set

4 Image Classification

4.1 Introduction

4.1.1 About This Experiment

This experiment is about a basic task in computer vision, that is, image recognition. The numay and TensorFlow frameworks are required. The NumPy arrays are used as the image objects. The TensorFlow framework is mainly used to create deep learning algorithms and build a convolutional neural network (CNN). This experiment recognizes image categories based on the CIFAR10 dataset.

4.1.2 Objectives

- Upon completion of this task, you will be able to:
  - Strengthen the understanding of Keras-based neural network model construction process
  - Master the method to load the pre-trained model.
  - Learn to use checkpoint function.
  - Master how to use a trained model to make predictions.

4.2 Experiment Code

4.2.1 Introducing Dependencies

Code:

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

4.2.2 Data Preprocessing

Code:

```
#download Cifar-10 dataset
(x_train,y_train), (x_test, y_test) = datasets.cifar10.load_data()
#print the size of the dataset
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
print(y_train[0])

#Convert the category label into onehot encoding
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)
```

Show the first 9 images

Code:

```
#Create a image tag list
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(x_train[i])
    #show the label
    plt.ylabel(category_dict[y_train[i][0]])
plt.show()
```

Output:

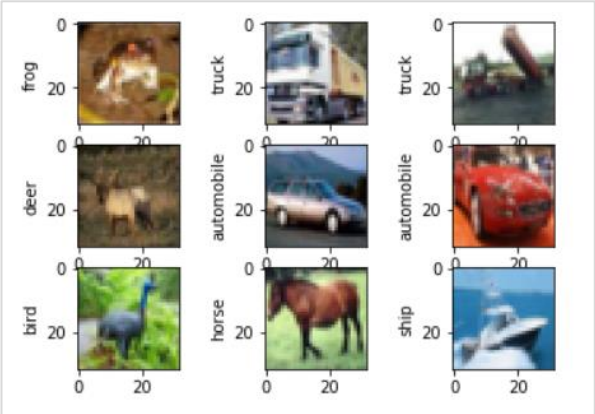


Figure 4-1 First 9 Images with Tags

Code:

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

4.2.3 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
```

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 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
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 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.

4.2.5 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 [==============================] - 13s 1ms/sample - loss: 0.8581 - accuracy:
0.7042s - loss: 0.854
[0.8581173644065857, 0.7042]
```

Predict on a single image.

Code:

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

Output:

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

Code:

```
#output the predicted label
new_model.predict_classes(x_test[1:2])
```

Output:

```
array([0], dtype=int64)
```

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:

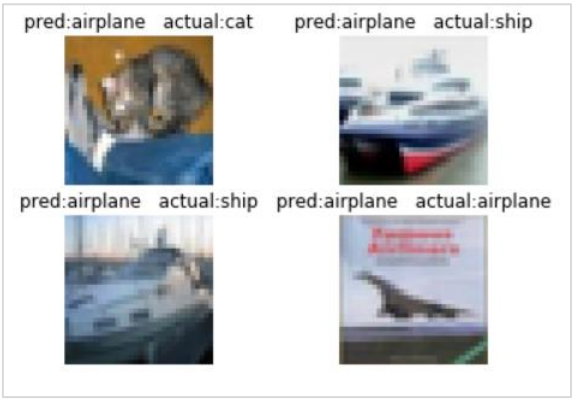


Figure 4-2 First 4 Images with Predicted Labels

4.3 Summary

This chapter describes how to build an image classification model based on TensorFlow 2 and python. It provides trainees with a basic concept of deep learning model building.

Huawei AI Certification Training

HCIA-AI

ModelArts Experiment Guide

ISSUE:3.0

HUAWEI TECHNOLOGIES CO., LTD.

Huawei Technologies Co., Ltd.

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

Website: http://e.huawei.com

Huawei Certificate System

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

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

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

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


Huawei Certification Portfolio
HUAWEI
Huawei Certification
ICT Vertical Certification
Finance
Public Safety
Platform and Service Certification
Big Data
AI
IoT
Intelligent Vision
Enterprise Communication
GaussDB
Kunpeng Application Developer
Cloud Computing
Cloud Service
ICT Infrastructure Certification
Data Center
Storage
Intelligent Computing
Security
Datacom
WLAN
SDN
Transmission
Access
LTE
5G
HCIE
Huawei Certified ICT Expert
HCIP
Huawei Certified ICT Professional
HCIA
Huawei Certified ICT Associate

About This Document

Overview

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

Description

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

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

Background Knowledge Required

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

Experiment Environment Overview

Huawei Cloud and ModelArts

Contents

1 ExeML

1.1 Introduction

1.1.1 About This Experiment

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

Image classification is based on image content labeling. An image classification model can predict a label corresponding to an image, and is applicable to scenarios in which image classes are obvious.

1.1.2 Objectives

This lab uses three specific examples to help you quickly create image classification, object detection, and predictive analytics models. The flower recognition experiment recognizes flower classes in images.

1.1.3 Experiment Environment Overview

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

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

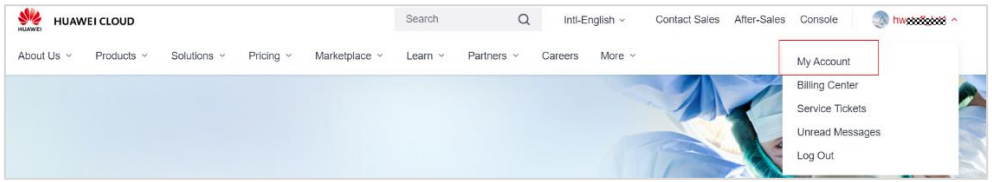


Figure 1-1 Huawei Cloud Homepage

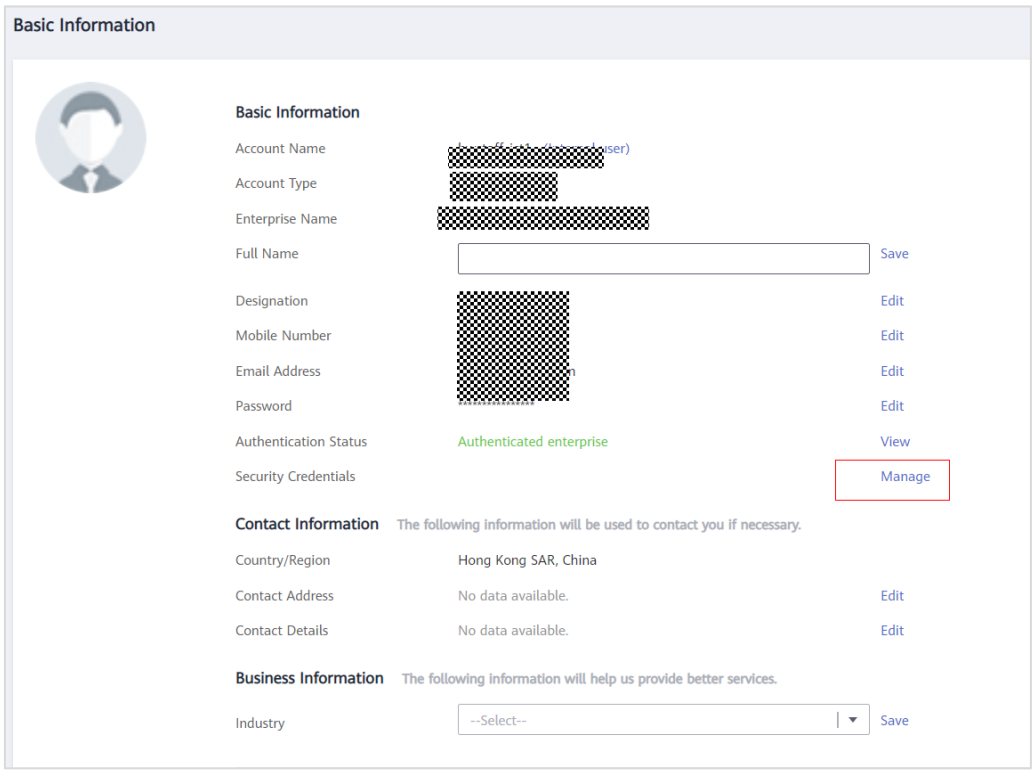


Figure 1-2 Basic Information Page

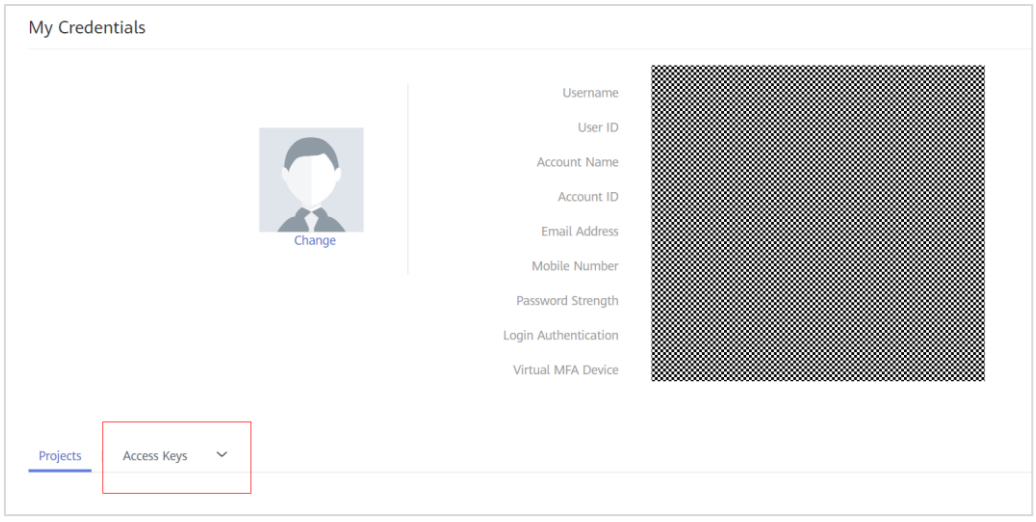


Figure 1-3 My Credentials Page

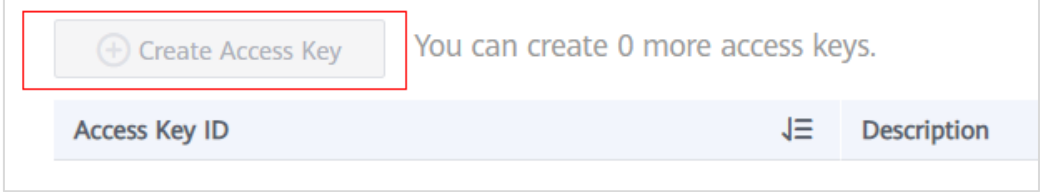


Figure 1-4 Create Access Key

- 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.

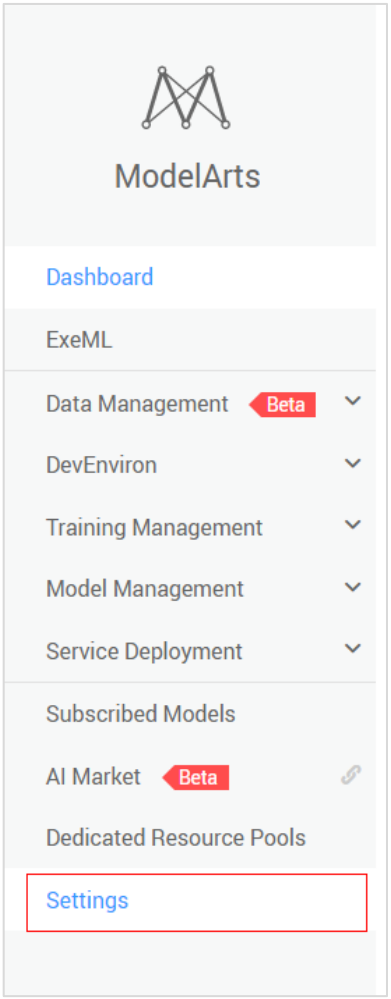


Figure 1-5 ModelArts Settings Page

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.

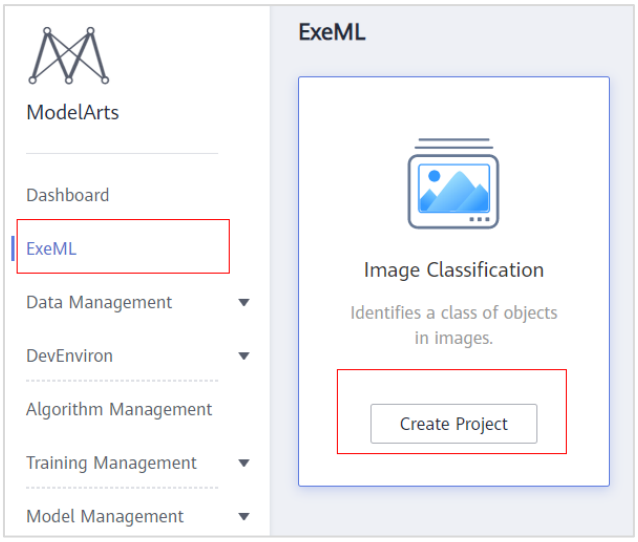


Figure 1-6 ModelArts ExeML

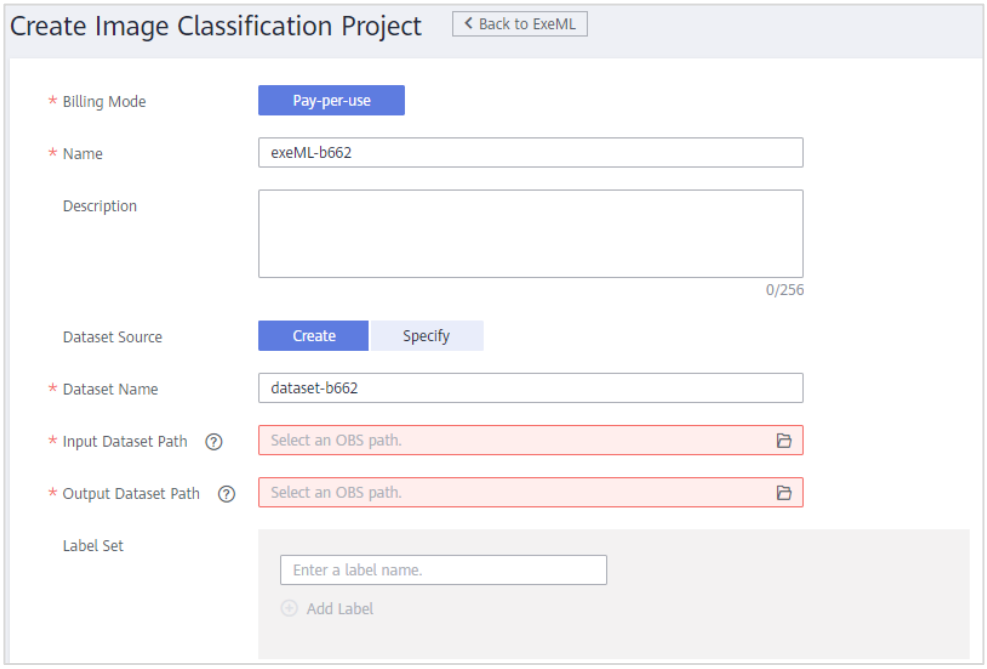


Figure 1-7 Create Image Classification Project

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.

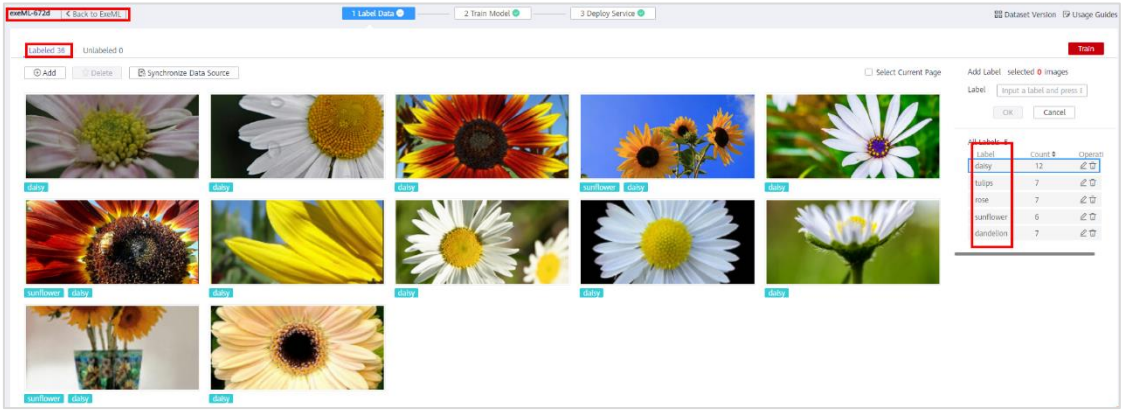


Figure 1-8 Upload Images

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.

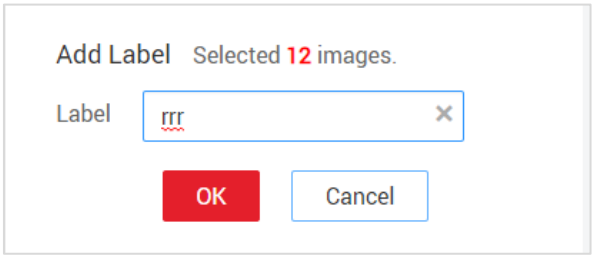


Figure 1-9 Label the Images

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.

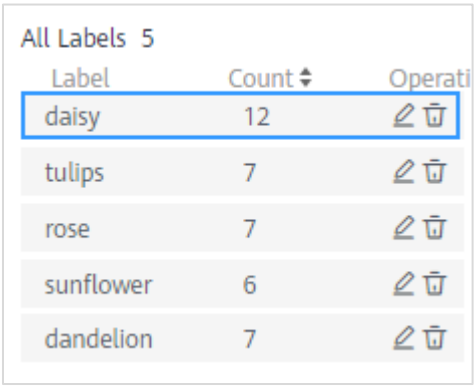


Figure 1-10 Delete or Modify Label in One Image

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.**

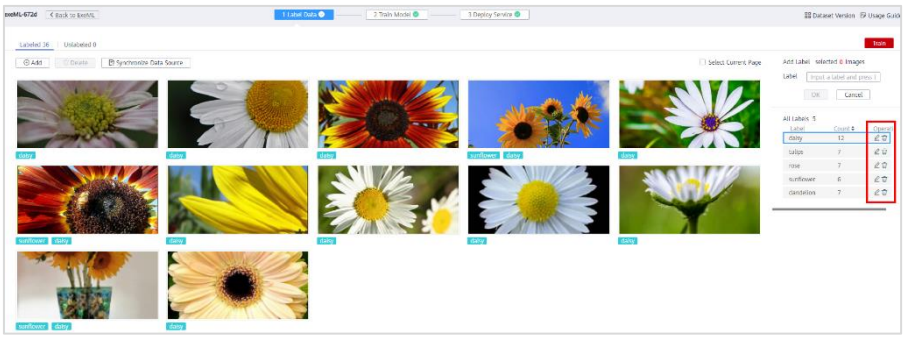


Figure 1-11 Delete or Modify a Label in Multiple Images

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.

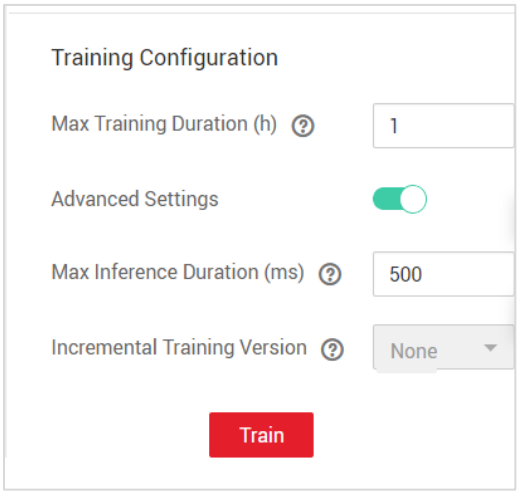


Figure 1-12 Training Configuration

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.

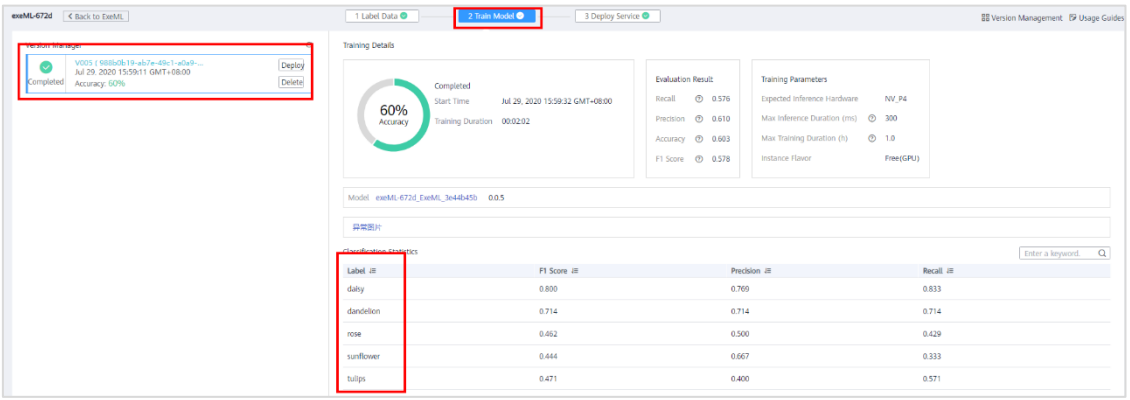


Figure 1-13 Train Model

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".

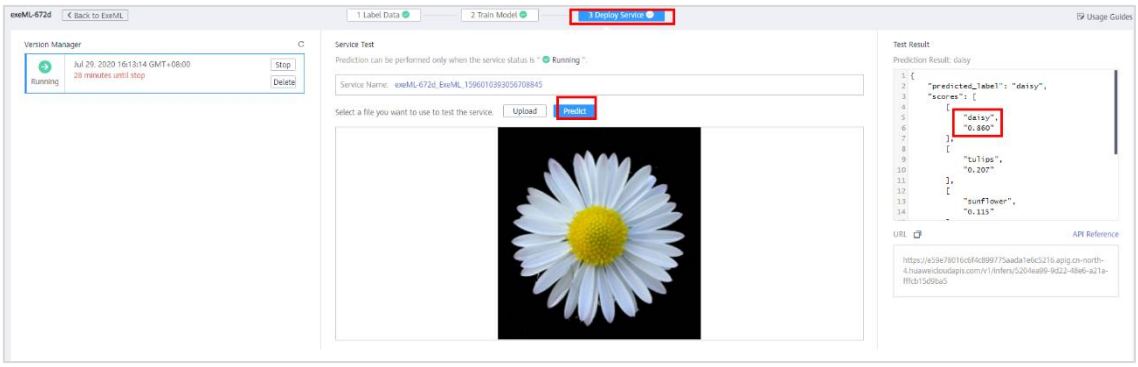


Figure 1-14 Deploy Service

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.

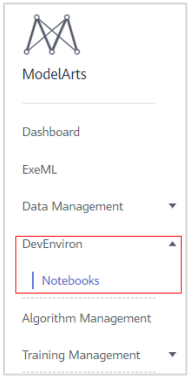


Figure 2-1 ModelArts Homepage

Click the Create button to Create the Notebook environment.

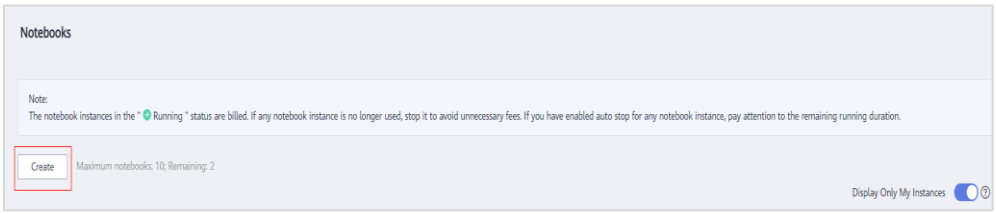


Figure 2-2 Create Notebook

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.

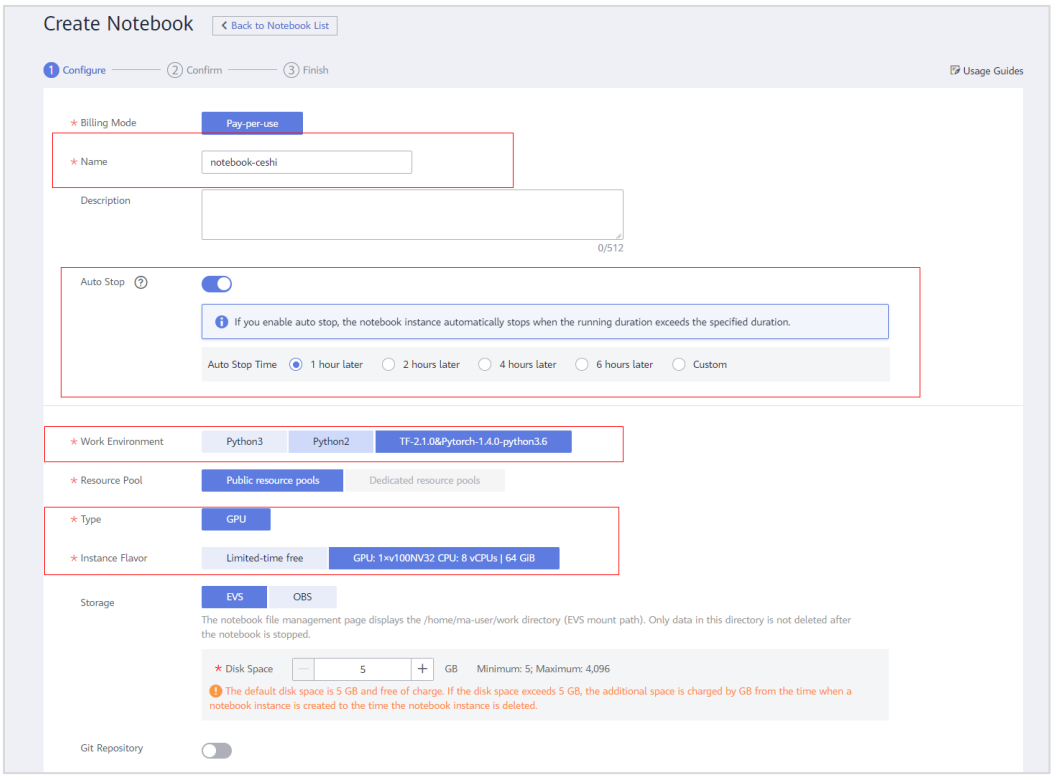


Figure 2-3 Notebook Configuration

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.

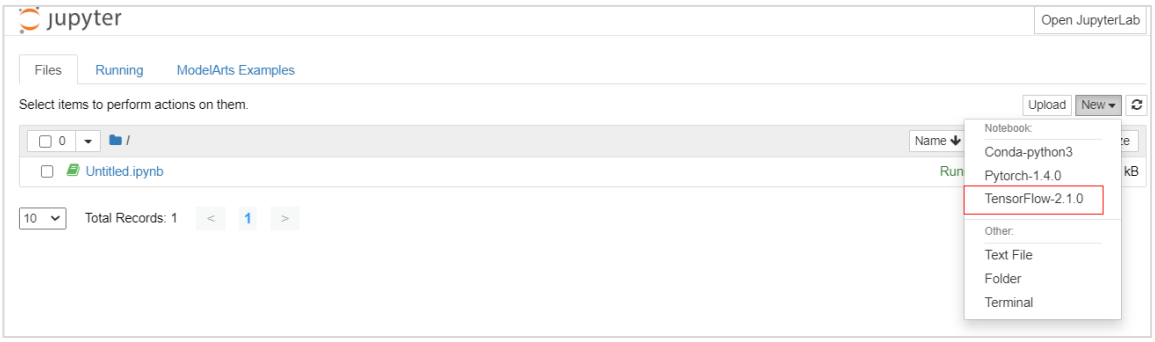


Figure 2-5 Create a New File

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

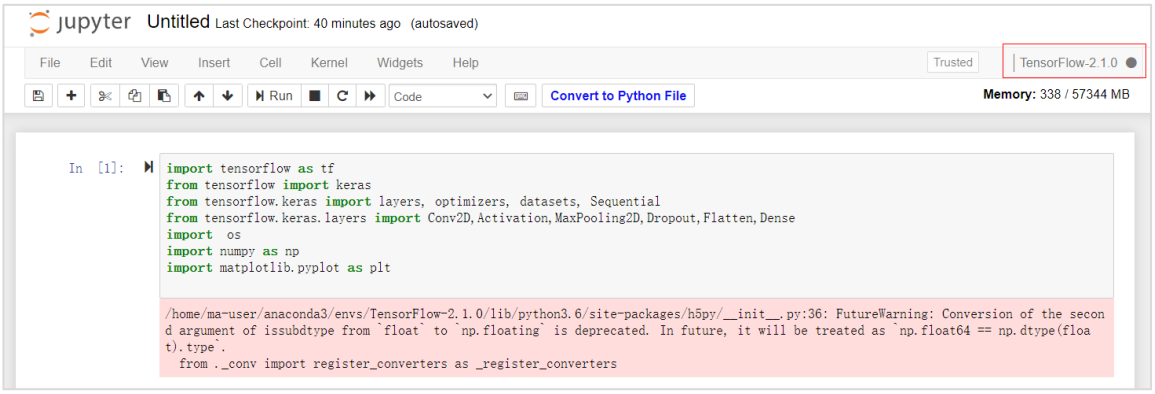


Figure 2-6 Notebook's Code Editing Interface

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


#download Cifar-10 dataset
(x_train,y_train), (x_test, y_test) = datasets.cifar10.load_data()
#print the size of the dataset
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
print(y_train[0])
```

Output:

```
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)
array([6], dtype=uint8)
```

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(Xtr['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:

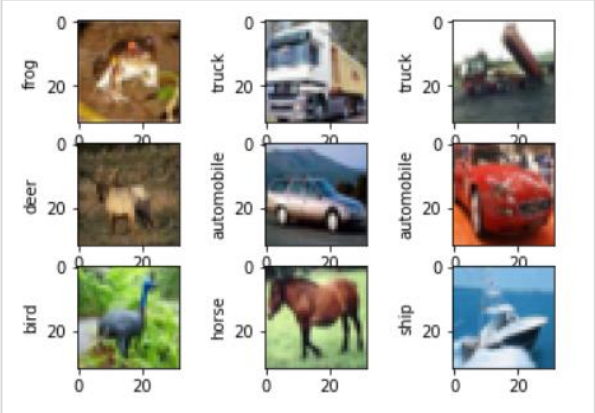


Figure 2-7 First 9 Images with Tags

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 [==============================] - 5s 508us/sample - loss: 125.2556 - accuracy: 0.2193
[125.25564072265625, 0.2193]
```

Predict on a single image.

Code:

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

Output:

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

Code:

```
#output the predicted label
new_model.predict_classes(x_test[1:2])
```

Output:

```
array([0], dtype=int64)
```

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.