

华为认证 AI 系列教程

HCIP-AI-EI Developer

华为企业 AI 开发高级工程师认证

图像处理

实验指导手册

版本：2.5



华为技术有限公司

版权所有 © 华为技术有限公司 2026。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<https://e.huawei.com>

华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT基础设施认证、基础软硬件认证、云平台及云服务认证三类认证。

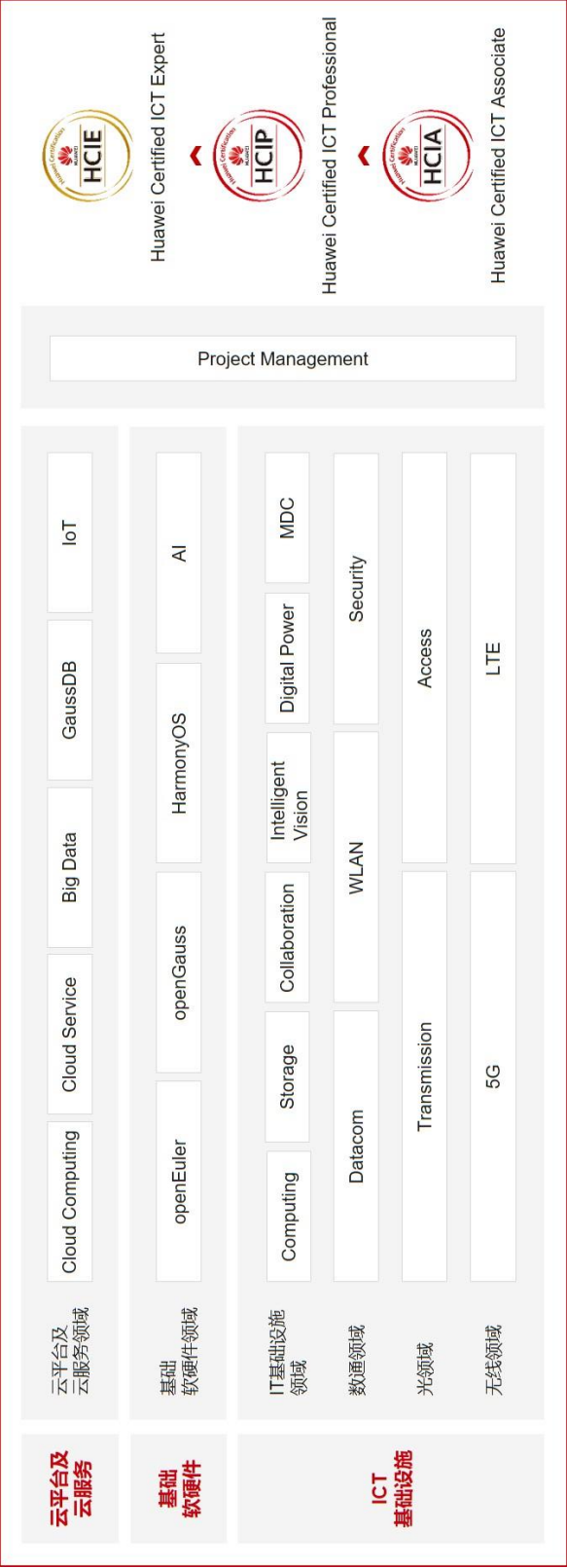
根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIP-AI-EI Developer V2.5定位于培养与认证具有图像处理、语音处理、自然语言处理能力，能够应用通用开源框架和华为云ModelArts平台等进行开发和创新的高级工程师。

通过HCIP-AI-EI Developer V2.5认证，您将掌握神经网络基础、图像处理理论、语音处理理论、自然语言处理理论、大模型深度学习算法等基础知识，具备运用通用开源框架和华为云ModelArts平台等进行图像处理、语音处理和自然语言处理等人工智能产品设计开发的能力，能够胜任AI开发高级工程师、图像处理开发工程师、语音处理开发工程师、自然语言处理开发工程师等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在人工智能世界的潮头浪尖！



简介

本书为 HCIP-AI-EI Developer V2.5 认证培训教程，适用于准备参加 HCIP-AI-EI Developer V2.5 考试的学员或者希望了解 AI 基础知识的读者。掌握本实验手册内容，您将能够了解如何进行图像文件的预处理，利用华为云实现图像标签功能的实际开发，自定义实现卷积神经网络完成图像分类。图像预处理的实验，我们使用了 OpenCV 库进行实现。图像标签的实验操作主要是通过提交 Restful 请求调用华为公有云服务中的相关服务，华为企业云服务为大家提供了丰富的图像处理应用的接口。卷积神经网络图像分类的实验使用了 PyTorch 开发框架进行实现。

内容描述

本实验指导书共包含 3 组实验，涉及到基于 Python 语言的 OpenCV 图像预处理实验，基于华为云的图像标签任务的智慧相册实验提升 AI 图像处理的实践开发能力，基于 PyTorch 构建卷积神经网络模型实现图像分类。

- 实验 1 图像数据预处理
- 实验 2 华为云图像标签实现智慧相册
- 实验 3 使用卷积神经网络实现图像分类

读者知识背景

本课程为华为认证开发课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基本的 Python 编程能力
- 有一定的数据结构基础

实验环境说明

- 华为云 ModelArts 服务（推荐）
-

准备实验环境

步骤 1 进入华为云 ModelArts 控制台

在[华为云 ModelArts 主页](https://www.huaweicloud.com/product/modelarts.html)（<https://www.huaweicloud.com/product/modelarts.html>），点击“控制台”进入 ModelArts 的管理页面。

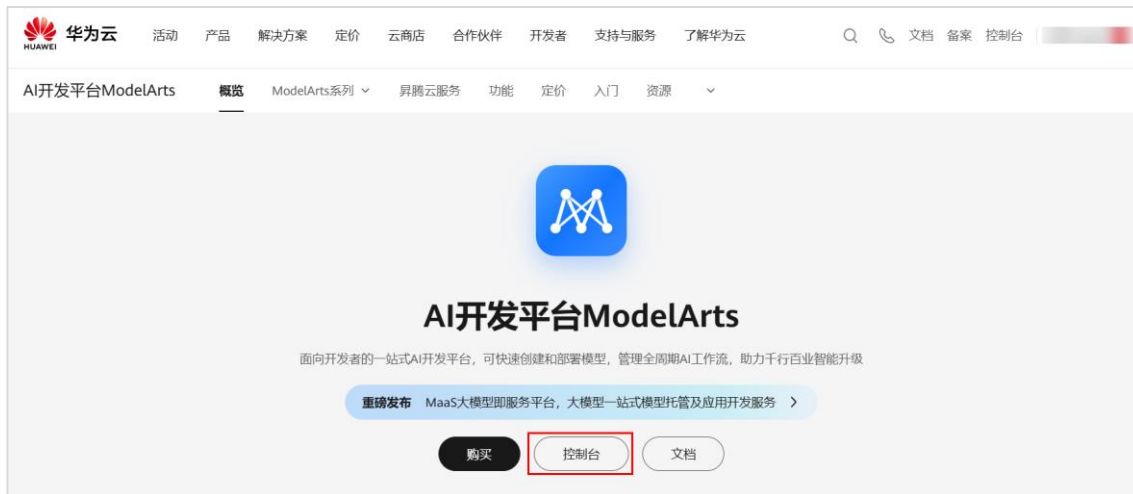


图1-1 华为云 ModelArts 主页

步骤 2 创建 Notebook 训练作业

在左侧菜单栏中选择“开发空间”的“Notebook”，点击进入 Notebook 创建页面。

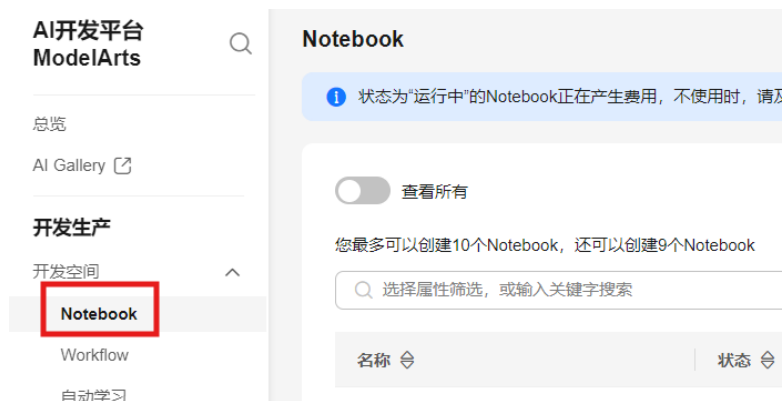


图1-2 ModelArts 控制台

点击“创建 Notebook”按钮，创建一个新的 Notebook，其配置如下：

- 名称：自定义
- 自动停止：建议 2 小时
- 镜像：pytorch2.5_cuda11.8
- 资源池：公共资源池
- 类型：GPU
- 规格：GPU: 1*Vnt1(16GB)|CPU: 8vCPUs 64GB

- 存储配置：云硬盘 EVS
- 磁盘规格：20GB

如图所示：

Figure 1-3 shows the configuration page for a Notebook instance. The configuration includes:

- Name:** notebook-hcip-image
- Description:** (Empty text box)
- Auto Stop:** Enabled (toggle switch). A message states: "After enabling this option, the Notebook instance will automatically stop after exceeding the selected runtime duration." Duration options: 1 hour, 2 hours (selected), 4 hours, 6 hours, Custom.
- Image:** Public Image (selected). Search bar: "请输入镜像名称". Table of images:

名称	版本	所属组织	描述
pytorch2_5_cuda11.8	v1	ai-certification	pytorch2.5.1+GPU
pytorch2_5_cuda11.8	1.1	ai-certification	--
- Resource Type:** Public Resource Pool (selected), Dedicated Resource Pool.
- Type:** GPU (selected).
- Instance Specification:** GPU: 1*Vnt1(16GB)|CPU: 8vCPUs 64GB (selected). Description: "One GP Vnt1 GPU with 16GB of memory, ideal for deep learning algorithm training and debugging."
- Storage Configuration:** Cloud Disk EVS (selected). Description: "云硬盘EVS作为持久化存储挂载在/home/ma-user/work目录下, 该目录下的内容在实例停止后会被保留, 存储支持在线按需扩容". Disk size: 20 GB (selected). Note: "磁盘规格默认为5GB, 从Notebook实例创建成功起, 直至删除成功, 每GB按照规定费用收费。"
- SSH Remote Development:** Disabled (toggle switch).

图1-3 Notebook 配置

配置完成之后“立即创建”，规格确认无误之后“提交”。

步骤 3 启动 Notebook 进入开发环境

当上一步创建好 Notebook 状态显示为“运行中”时，在右侧“操作”栏中“打开”，即可进入在线编程页面。

Figure 1-4 shows the Notebook management page. The table lists the created Notebook instances:

名称	状态	镜像	实例规格	描述	创建时间	创建者	操作
notebook-8f0c 16824816-37b7-42f4-952e-9fca1e789668	运行中 (59%)	pytorch2_5_cuda11.8.1.1	GPU: 1*Vnt1(...)	--			打开 启动 停止 更多

图1-4 打开 Notebook 环境

可以在此页面创建或编辑项目，如图所示：

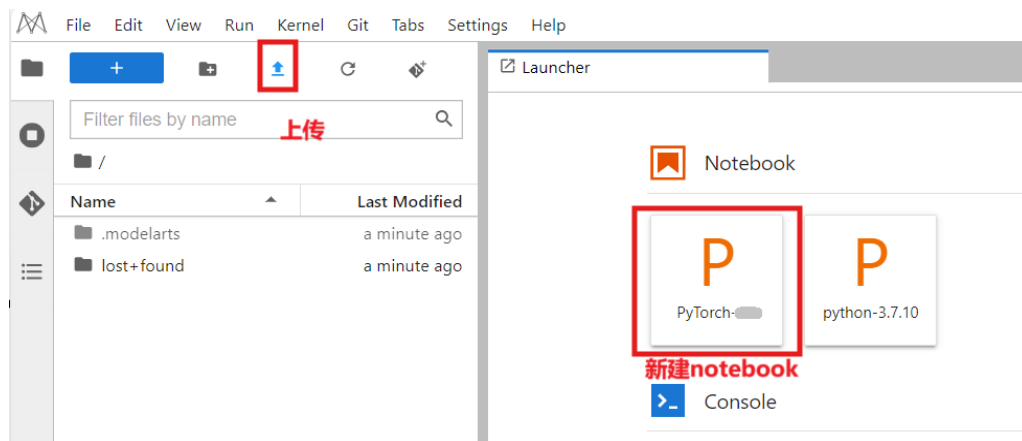


图1-5 页面操作

*注意：Notebook 环境内上传、创建和编辑的文件均在/home/ma-user/work 目录下。

步骤 4 停止 Notebook 训练作业

实验完成之后，请及时关闭 Notebook 训练作业，避免产生不必要的资源浪费。

登录华为云 ModelArts 控制台，在“操作”栏选择“停止”操作。

如下图所示：



图1-6 及时停止 Notebook

至此，训练用的线上 Notebook 环境搭建完成。

目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
实验环境说明	3
准备实验环境	4
1 图像数据预处理	9
1.1 实验简介	9
1.2 实验目的	9
1.3 实验环境说明	9
1.4 实验步骤	9
1.4.1 图像基本操作	9
1.4.2 图像颜色空间转换	11
1.4.3 图像几何变换	12
1.4.4 图像灰度变换	18
1.4.5 图像形态学处理	23
1.4.6 图像滤波	25
1.5 实验小结	30
1.6 思考题	30
2 华为云图像标签实现智慧相册	31
2.1 实验简介	31
2.2 实验目的	31
2.3 实验步骤	31
2.3.1 申请服务	32
2.3.2 通过 AK/SK 进行图像标签服务（如已有，可跳过）	33
2.3.3 打开 Jupyter Notebook	35
2.3.4 安装 Python 版本 SDK	35
2.3.5 图像标签服务初始化	35
2.3.6 利用标签打标相关的照片	37
2.3.7 利用打标结果制作动态相册	39
2.3.8 利用标签对照片进行自动分类	41



2.4 实验小结	42
2.5 思考题	42
3 使用卷积神经网络实现图像分类	43
3.1 实验简介	43
3.2 实验目的	43
3.3 实验步骤	43
3.3.1 数据预处理	43
3.3.2 构建卷积神经网络	46
3.3.3 模型训练	48
3.3.4 测试网络模型	50
3.4 实验小结	52
3.5 思考题	52

1

图像数据预处理

1.1 实验简介

图像预处理的主要目的是消除图像中无关的信息，恢复有用的真实信息，增强有关信息的可检测性、最大限度地简化数据，从而改进特征提取、图像分割、匹配和识别的可靠性。

本实验使用 OpenCV 图像处理库实现基本的图像预处理操作。包括图像的基本操作、颜色空间转换、几何变换、灰度变换、形态学处理和图像滤波。

1.2 实验目的

本实验将理论教材中介绍的图像预处理技术通过 Python 语言的 OpenCV 图像处理库实现。通过本实验学员将了解如何使用 OpenCV 进行图像预处理。通过实际图像数据的变化，加深对图像预处理技术的感知，本实验将指导学员理解和掌握如何使用 Python 语言进行图像预处理开发的方法和技巧。

1.3 实验环境说明

本实验建议安装 3.6 以上版本的 Python 环境，并且需要安装 OpenCV，numpy，matplotlib 第三方库。

数据链接：<https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIP-AI%20EI%20Developer/V2.5/chapter2/cv.zip>

1.4 实验步骤

1.4.1 图像基本操作

注意：所有实验 1.4 中代码所读取的图片，学员可选用自己电脑本地图片上传 Notebook。

步骤 1 图像的读取和显示

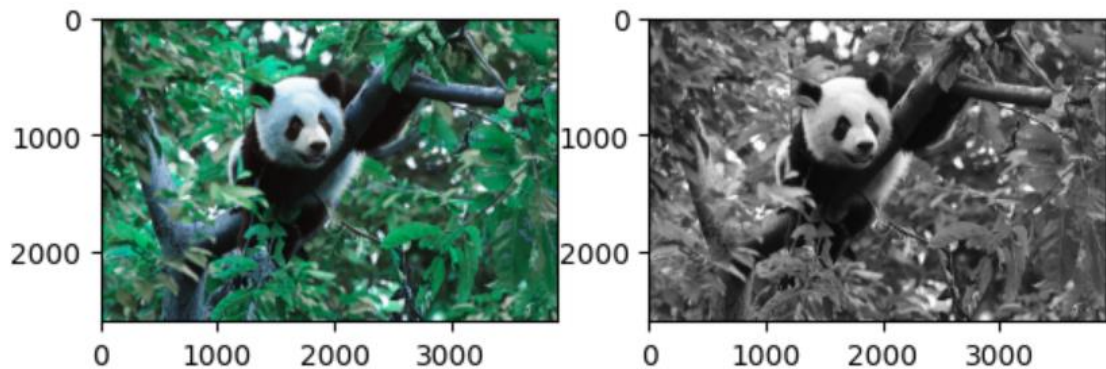
代码：

```
import cv2
import matplotlib.pyplot as plt
```

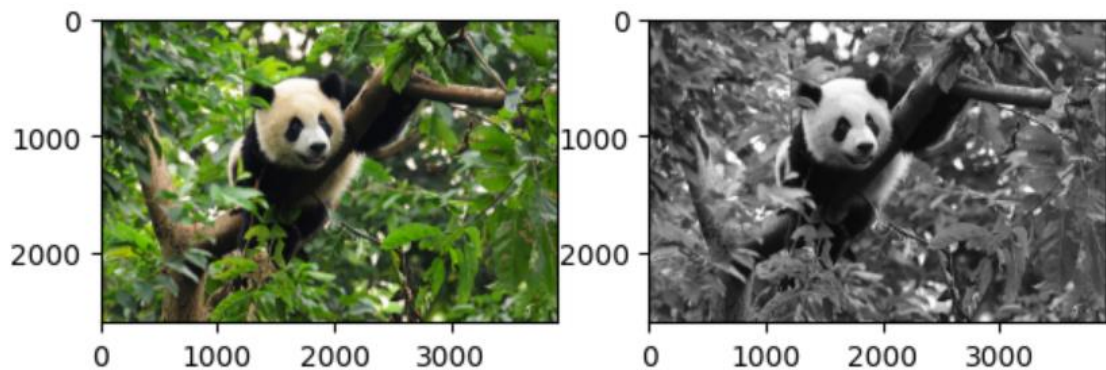
```
%matplotlib inline

# 读取图像，第二个参数为 1 表示读取彩色图像，为 0 表示图像灰度图像
im = cv2.imread(r"panda.jpg",1)
im_gray = cv2.imread(r"panda.jpg",0)
#新建画布
plt.figure()
#用 matplotlib 图片显示彩色不正常
plt.subplot(1,2,1),plt.imshow(im)
plt.subplot(1,2,2),plt.imshow(im_gray,'gray')
plt.show()
#因为 CV2 读取图像的通道顺序为 BGR，而 Matplotlib 显示图像的通道顺序为 RGB，因此需要调换颜色通道顺序
im_RGB = cv2.cvtColor(im,cv2.COLOR_BGR2RGB)
plt.subplot(1,2,1),plt.imshow(im_RGB)
plt.subplot(1,2,2),plt.imshow(im_gray,'gray')
```

结果：



(<AxesSubplot:~>, <matplotlib.image.AxesImage at 0x7fe831619d90>)



步骤 2 显示数据类型和图像尺寸

代码：

```
#打印图像数据类型
print(im.dtype)
# 打印图像大小
```

```
print(im.shape)
```

结果：

```
uint8
```

```
(2600, 3914, 3)
```

步骤 3 图像的保存

代码：

```
# 保存图像到指定路径
cv2.imwrite('panda.png',im)
```

结果：

```
True
```

1.4.2 图像颜色空间转换

步骤 1 彩色图像灰度化

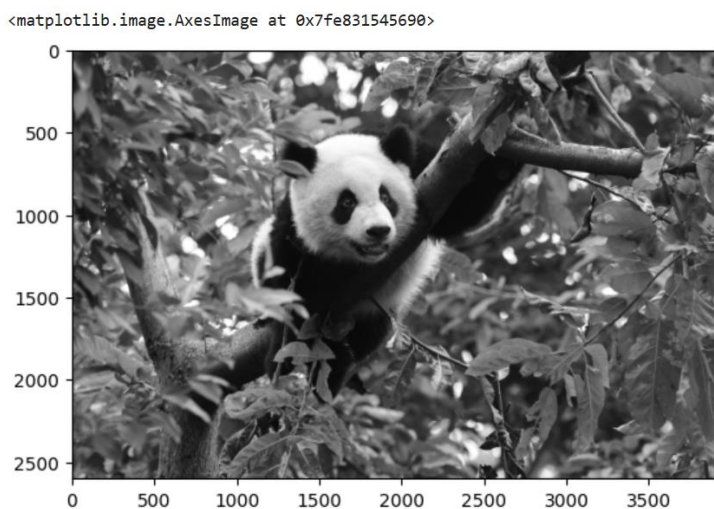
代码：

```
im = cv2.imread(r"panda.jpg")

# 使用 cvtColor 进行颜色空间变化 cv2.COLOR_BGR2GRAY 代表 BGR to gray
img_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)

plt.imshow(img_gray,'gray')
```

结果：



步骤 2 更换三通道顺序 BGR 为 HSV

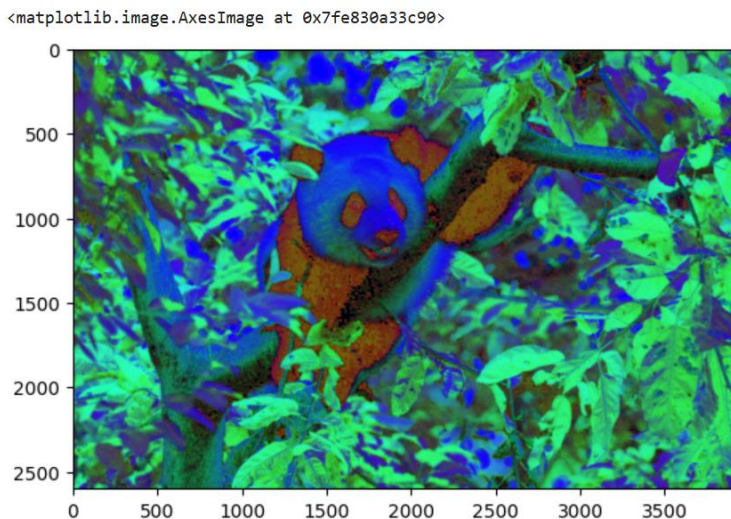
代码：

```
im = cv2.imread(r"panda.jpg")
```

```
# 使用 cvtColor 进行颜色空间变化 cv2.COLOR_BGR2HSV
im_hsv= cv2.cvtColor(im, cv2.COLOR_BGR2HSV)

# 当图像数据为 3 通道时, imshow 函数认为数据是 RGB 的
# 使用 imshow 显示 HSV 数据, 会发现图片显示颜色畸变
plt.imshow(im_hsv)
```

结果:



1.4.3 图像几何变换

步骤 1 平移

代码:

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

# 定义平移 translate 函数
def translate(img, x, y):
    # 获取图像尺寸
    (h, w) = img.shape[:2]

    # 定义平移矩阵
    M = np.float32([[1, 0, x], [0, 1, y]])

    # 使用 OpenCV 仿射变换函数实现平移操作
    shifted = cv2.warpAffine(img, M, (w, h))

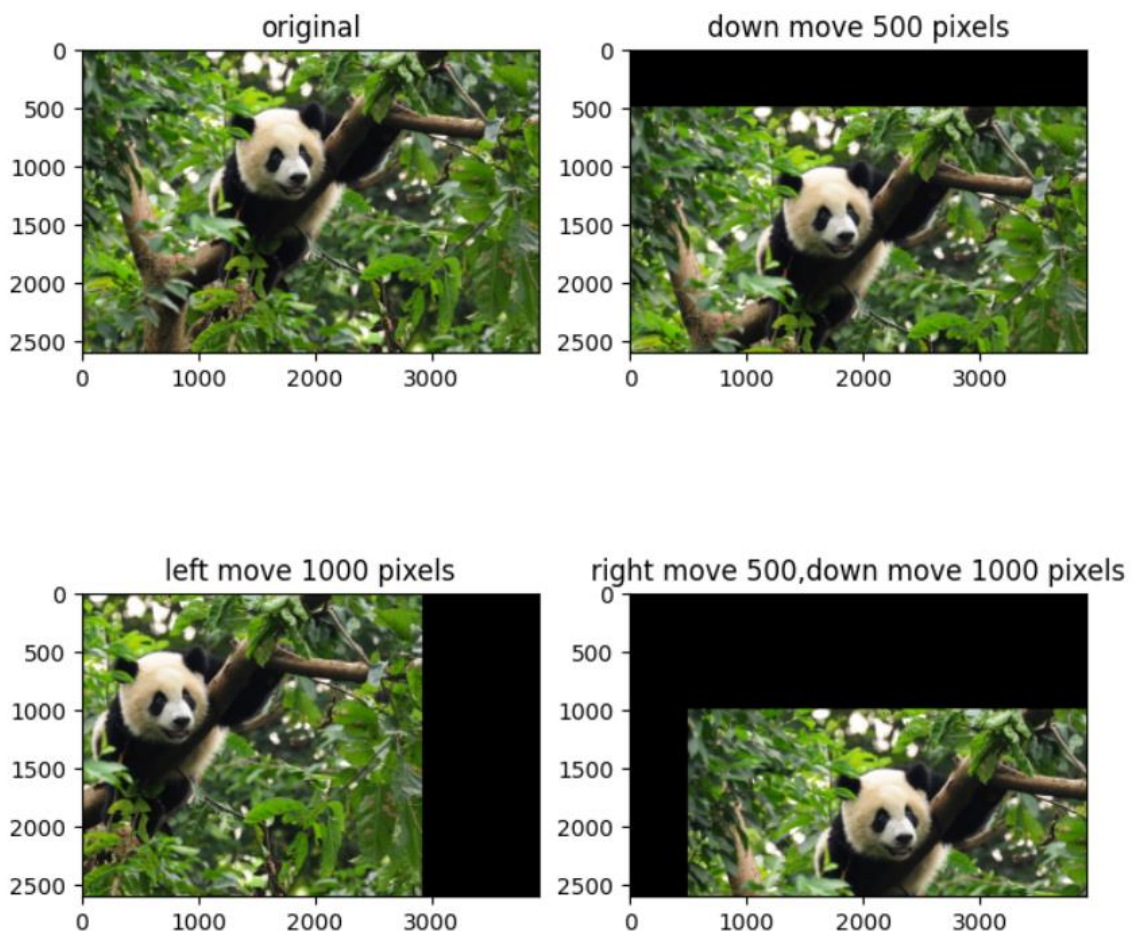
    # 返回转换后的图像
    return shifted
```

```
# 加载图像并显示
im = cv2.imread('panda.jpg')
im_RGB = cv2.cvtColor(im,cv2.COLOR_BGR2RGB)
plt.figure(figsize=(8,8))
plt.subplot(2,2,1),plt.title('original'),plt.imshow(im_RGB)

# 对原图做平移操作
# 下移 500 像素
shifted_1 = translate(im_RGB, 0, 500)
plt.subplot(2,2,2),plt.title('down move 500 pixels'),plt.imshow(shifted_1)
# 左移 1000 像素
shifted_2 = translate(im_RGB, -1000, 0)
plt.subplot(2,2,3),plt.title('left move 1000 pixels'),plt.imshow(shifted_2)
# 右移 500, 下移 1000 像素
shifted_3 = translate(im_RGB, 500, 1000)
plt.subplot(2,2,4),plt.title('right move 500,down move 1000 pixels'),plt.imshow(shifted_3)
```

结果:

```
(<AxesSubplot:title={'center':'right move 500,down move 1000 pixels'}>,
 Text(0.5, 1.0, 'right move 500,down move 1000 pixels'),
 <matplotlib.image.AxesImage at 0x7fe830418c90>)
```



步骤 2 旋转

代码：

```
# 定义旋转 rotate 函数
def rotate(img, angle, center=None, scale=1.0):
    # 获取图像尺寸
    (h, w) = img.shape[:2]

    # 旋转中心的缺失值为图像中心
    if center is None:
        center = (w / 2, h / 2)

    # 调用计算旋转矩阵函数
    M = cv2.getRotationMatrix2D(center, angle, scale)

    # 使用 OpenCV 仿射变换函数实现旋转操作
    rotated = cv2.warpAffine(img, M, (w, h))

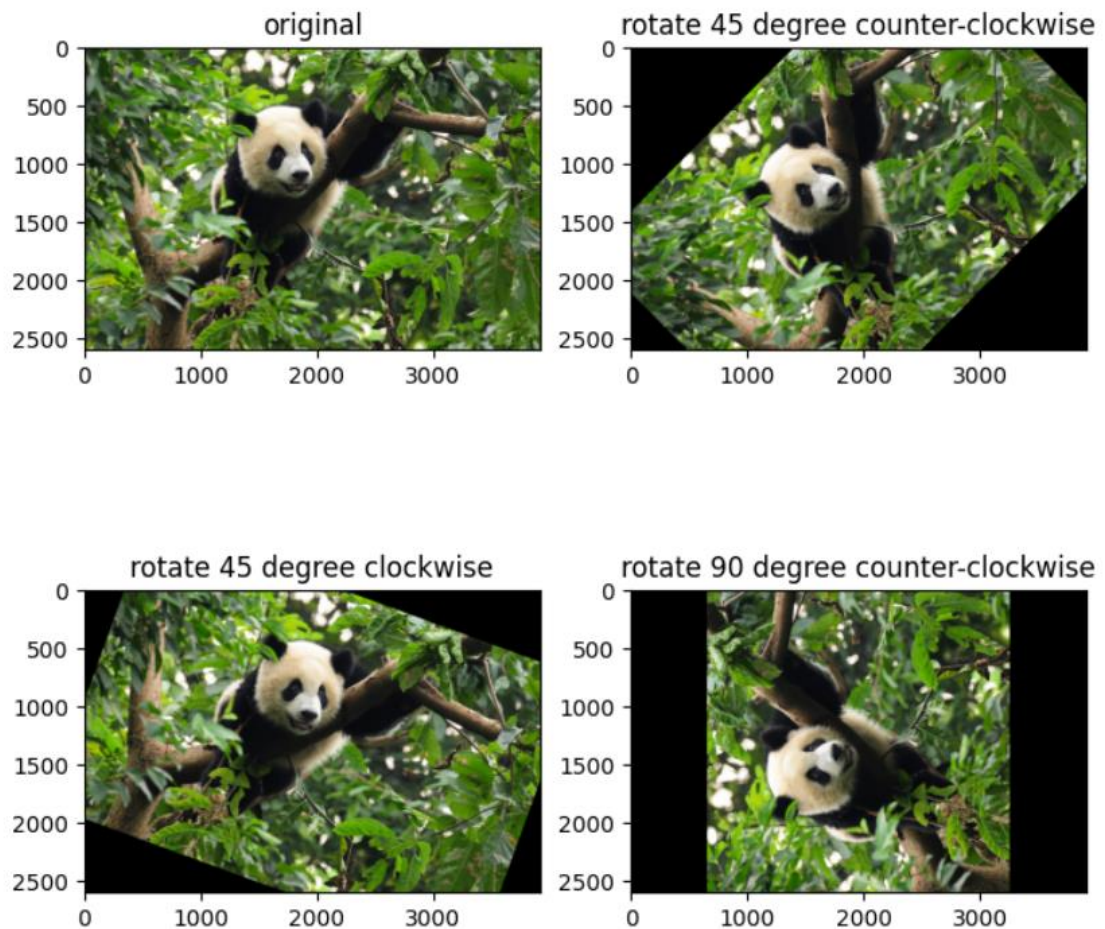
    # 返回旋转后的图像
    return rotated

im = cv2.imread('panda.jpg')
im_RGB = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(8,8))
plt.subplot(2,2,1), plt.title('original'), plt.imshow(im_RGB)

# 对原图做旋转操作
# 逆时针 45 度
rotated_45 = rotate(im_RGB, 45)
plt.subplot(2,2,2), plt.title('rotate 45 degree counter-clockwise'), plt.imshow(rotated_45)
# 顺时针 20 度
rotated_minus20 = rotate(im_RGB, -20)
plt.subplot(2,2,3), plt.title('rotate 45 degree clockwise'), plt.imshow(rotated_minus20)
# 逆时针 90 度
rotated_90 = rotate(im_RGB, 90)
plt.subplot(2,2,4), plt.title('rotate 90 degree counter-clockwise'), plt.imshow(rotated_90)
```

结果：

```
(<AxesSubplot:title={'center':'rotate 90 degree counter-clockwise'}>,
Text(0.5, 1.0, 'rotate 90 degree counter-clockwise'),
<matplotlib.image.AxesImage at 0x7fe83025be90>)
```



步骤 3 镜像

代码:

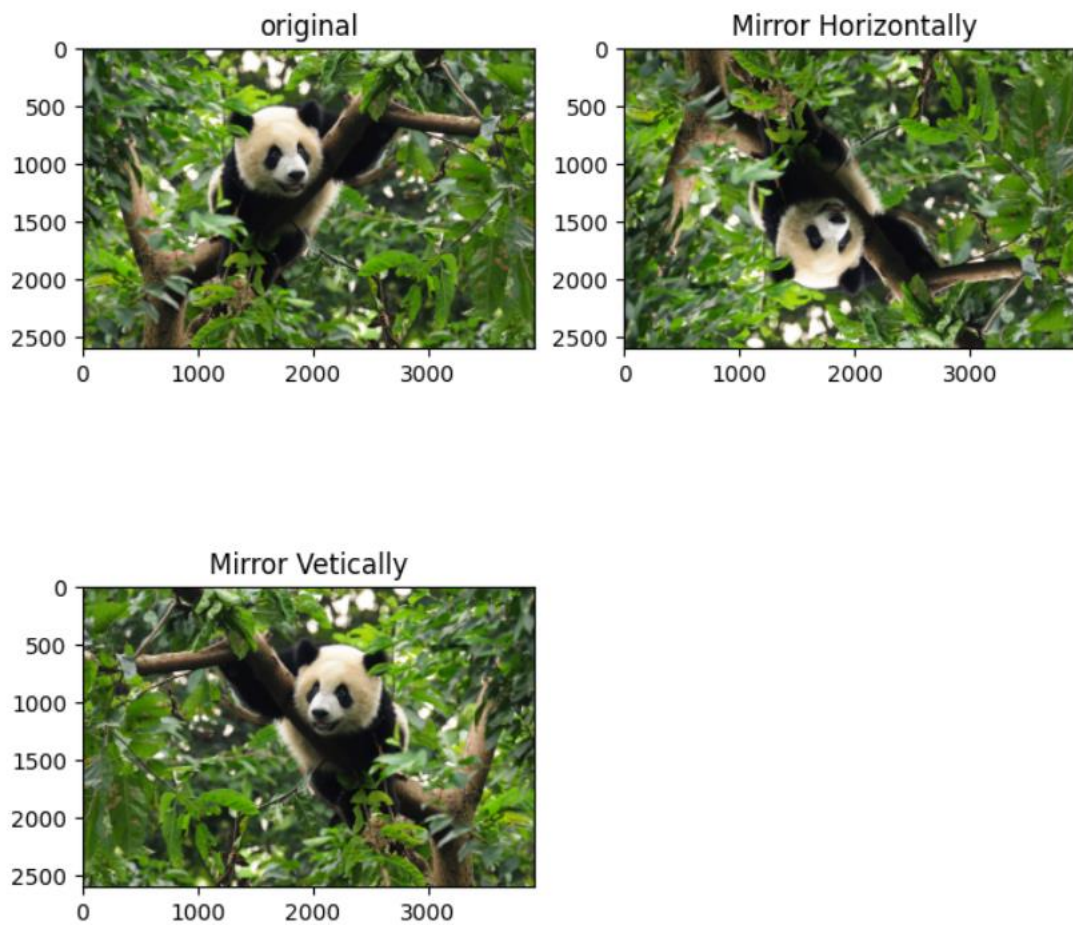
```
im = cv2.imread('panda.jpg')
im_RGB = cv2.cvtColor(im,cv2.COLOR_BGR2RGB)
plt.figure(figsize=(8,8))
plt.subplot(2,2,1),plt.title('original'),plt.imshow(im_RGB)

# 进行水平镜像
im_flip0 = cv2.flip(im_RGB, 0)
plt.subplot(2,2,2),plt.title(' Mirror Horizontally'),plt.imshow(im_flip0)

# 进行垂直镜像
im_flip1 = cv2.flip(im_RGB, 1)
plt.subplot(2,2,3),plt.title('Mirror Vetically'),plt.imshow(im_flip1)
```

结果:

```
(<AxesSubplot:title={'center':'Mirror Vetically'}>,
Text(0.5, 1.0, 'Mirror Vetically'),
<matplotlib.image.AxesImage at 0x7fe83012d810>)
```



步骤 4 缩放

代码：

```
im = cv2.imread('panda.jpg')
im_RGB = cv2.cvtColor(im,cv2.COLOR_BGR2RGB)
plt.figure(figsize=(8,8))
plt.subplot(2,2,1),plt.title('original'),plt.imshow(im_RGB)

# 获取图像尺寸
(h, w) = im_RGB.shape[:2]

# 缩放的目标尺寸
dst_size = (2000,3000)

# 最邻近插值
method = cv2.INTER_NEAREST

# 进行缩放
resized = cv2.resize(im_RGB, dst_size, interpolation = method)
```

```
plt.subplot(2,2,2),plt.title('INTER_NEAREST'),plt.imshow(resized)

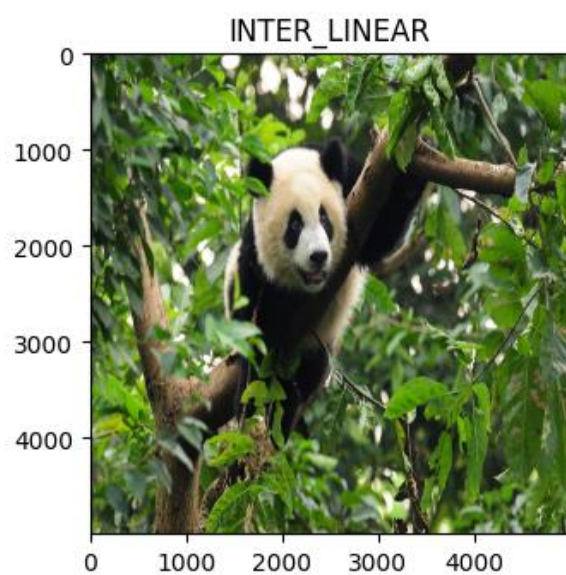
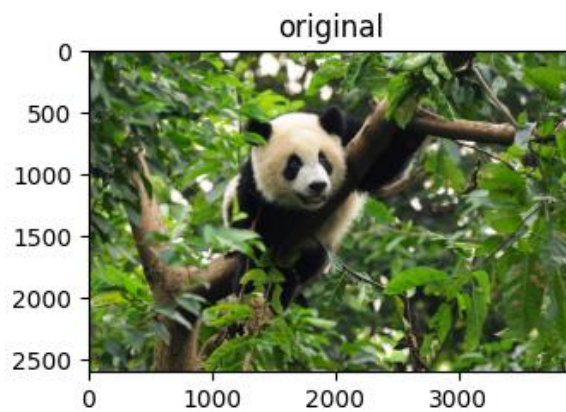
# 缩放的目标尺寸
dst_size = (5000,5000)

# 双线性插值
method = cv2.INTER_LINEAR

# 进行缩放
resized_linear = cv2.resize(im_RGB, dst_size, interpolation = method)
plt.subplot(2,2,3),plt.title('INTER_LINEAR'),plt.imshow(resized_linear)
```

结果：

```
(<AxesSubplot:title={'center':'INTER_LINEAR'}>,
 Text(0.5, 1.0, 'INTER_LINEAR'),
 <matplotlib.image.AxesImage at 0x7fe82fe79f10>)
```



1.4.4 图像灰度变换

步骤 1 反转，灰度拉伸，灰度压缩

代码：

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

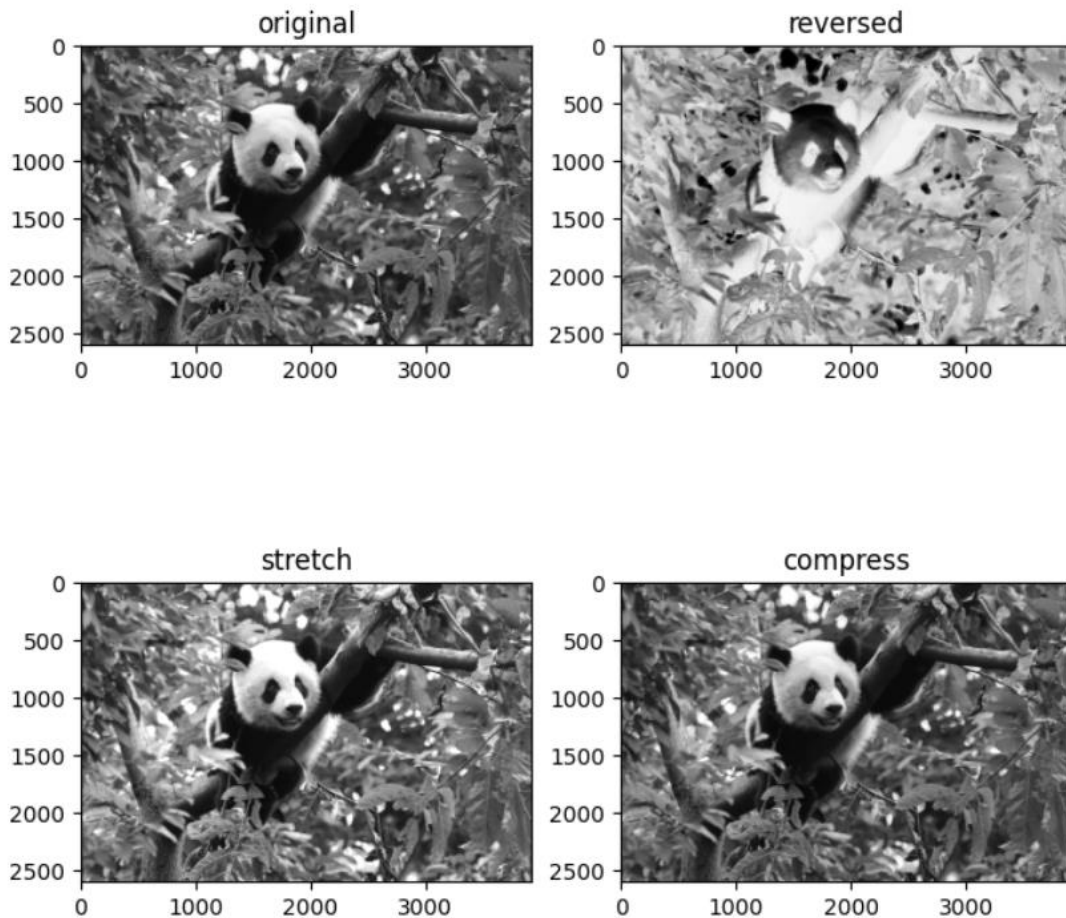
# 定义线性灰度变化函数
# k>1 时 实现灰度数值的拉伸
# 0<k<1 时 实现灰度数值的压缩
# k=-1 b=255 实现灰度反转
def linear_trans(img, k, b=0):
    # 计算灰度线性变化的映射表
    trans_list = [(np.float32(x)*k+b) for x in range(256)]
    # 将列表转换为 np.array
    trans_table = np.array(trans_list)
    # 将超过[0,255]灰度范围的数值进行调整,并指定数据类型为 uint8
    trans_table[trans_table>255] = 255
    trans_table[trans_table<0] = 0
    trans_table = np.round(trans_table).astype(np.uint8)
    # 使用 OpenCV 的 look up table 函数修改图像的灰度值
    return cv2.LUT(img, trans_table)

im = cv2.imread(r'panda.jpg',0)
plt.figure(figsize=(8,8))
plt.subplot(2,2,1),plt.title('original'),plt.imshow(im,'gray')

# 反转
im_inversion = linear_trans(im, -1, 255)
plt.subplot(2,2,2),plt.title('reversed'),plt.imshow(im_inversion,'gray')
# 灰度拉伸
im_stretch = linear_trans(im, 1.2)
plt.subplot(2,2,3),plt.title('stretch'),plt.imshow(im_stretch,'gray')
# 灰度压缩
im_compress = linear_trans(im, 0.4)
plt.subplot(2,2,4),plt.title('compress'),plt.imshow(im_compress,'gray')
```

结果：

```
(<AxesSubplot:title={'center':'compress'}>,
Text(0.5, 1.0, 'compress'),
<matplotlib.image.AxesImage at 0x7fe82fc8c450>)
```



步骤 2 伽马变换

代码：

```
def gamma_trans(img, gamma):
    # 先归一化到 1，做伽马计算，再还原到[0,255]
    gamma_list = [np.power(x / 255.0, gamma) * 255.0 for x in range(256)]
    # 将列表转换为 np.array，并指定数据类型为 uint8
    gamma_table = np.round(np.array(gamma_list)).astype(np.uint8)
    # 使用 OpenCV 的 look up table 函数修改图像的灰度值
    return cv2.LUT(img, gamma_table)

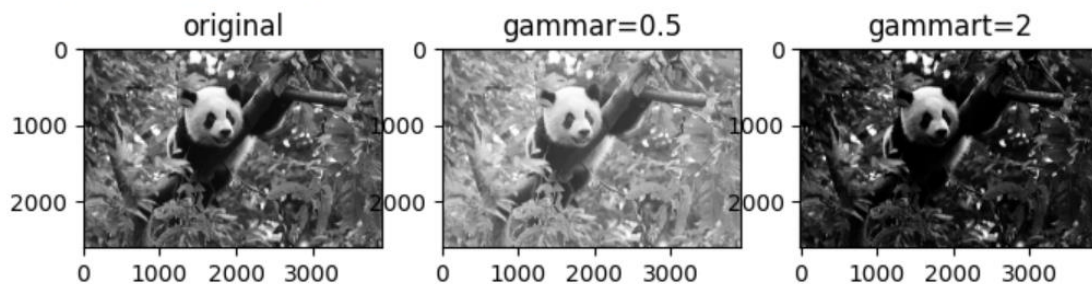
im = cv2.imread(r'panda.jpg',0)
plt.figure(figsize=(8,8))
plt.subplot(1,3,1),plt.title('original'),plt.imshow(im,'gray')

# 使用伽马值为 0.5 的变化，实现对暗部的拉升，亮部的压缩
im_gamma05 = gamma_trans(im, 0.5)
plt.subplot(1,3,2),plt.title('gamma=0.5'),plt.imshow(im_gamma05,'gray')
```

```
# 使用伽马值为 2 的变化，实现对亮部的拉升，暗部的压缩
im_gamma2 = gamma_trans(im, 2)
plt.subplot(1,3,3),plt.title('gammart=2'),plt.imshow(im_gamma2,'gray')
```

结果：

```
(<AxesSubplot:title={'center':'gammart=2'}>,
 Text(0.5, 1.0, 'gammart=2'),
 <matplotlib.image.AxesImage at 0x7fe82fb0f4d0>)
```



步骤 3 直方图均衡化

代码：

```
import cv2
from matplotlib import pyplot as plt

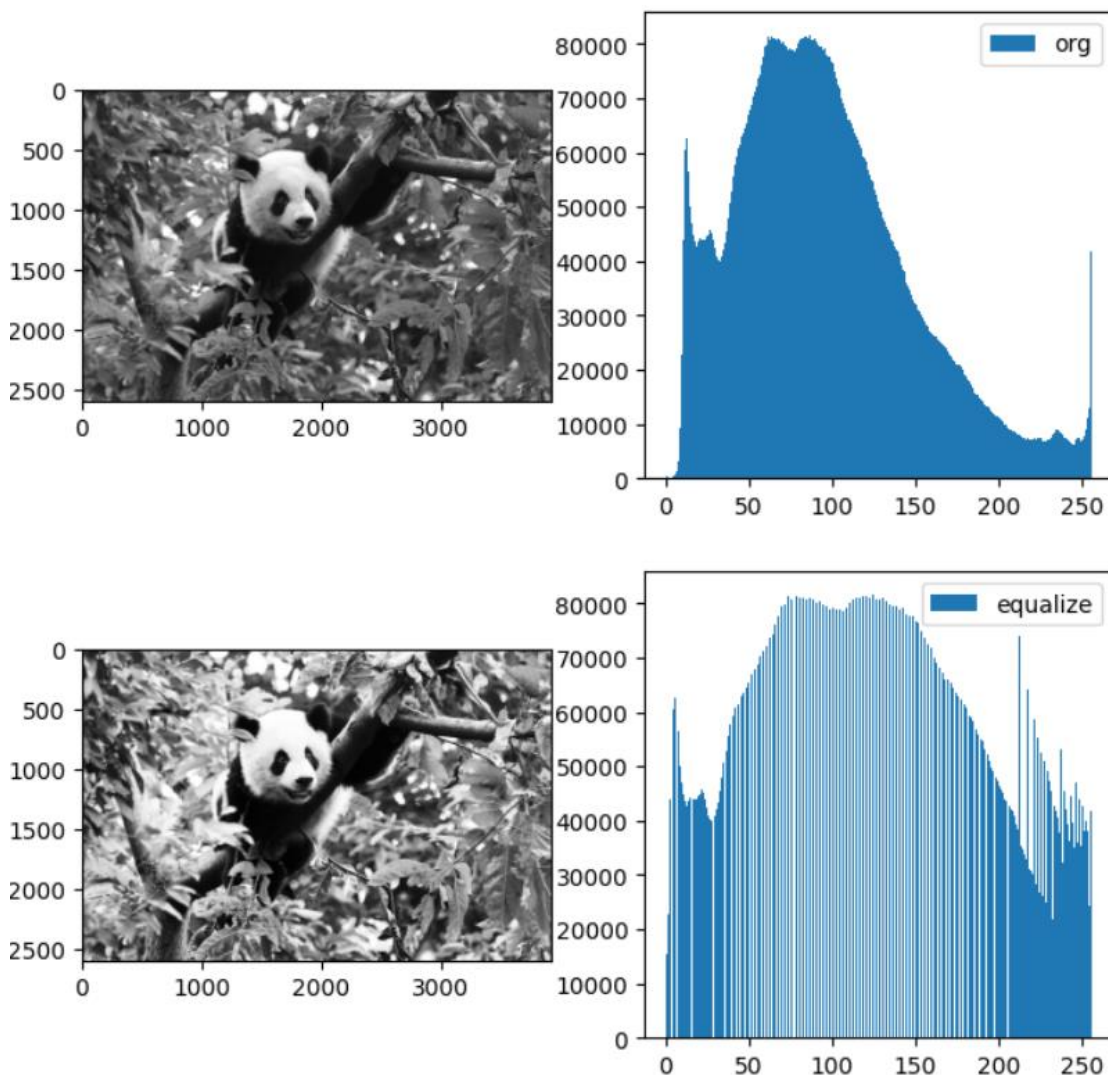
im = cv2.imread(r'panda.jpg',0)
plt.figure(figsize=(8,8))
plt.subplot(2,2,1),plt.imshow(im,'gray')

# 调用 OpenCV 的直方图均衡化 API
im_equa1 = cv2.equalizeHist(im)
plt.subplot(2,2,3),plt.imshow(im_equa1,'gray')

# 显示原始图像的直方图
plt.subplot(2,2,2)
plt.hist(im.ravel(), 256, [0,256],label='org')
plt.legend()

# 显示均衡化图像的直方图
plt.subplot(2,2,4)
plt.hist(im_equa1.ravel(), 256, [0,256],label='equalize')
plt.legend()
plt.show()
```

结果：



步骤 4 图像二值化

一幅图像包括前景（目标）、背景还有噪声，要想从数字图像中直接提取出前景目标，常用的方法就是二值法。

设定一个阈值 T ，用 T 将图像的数据分成两部分：大于 T 的像素群和小于 T 的像素群。这是研究灰度变换的方法，称为图像的二值化（Binarization）。

Python-OpenCV 中提供了阈值（threshold）函数：cv2.threshold（src,x,y,method）。

- src：原图像，原图像应该是灰度图
- x：用来对像素值进行分类的阈值
- y：填充色
- method：指选择二值的方法

该函数有两个返回值，第一个 retVal 是得到的阈值，第二个就是阈值化后的图像。

代码：

```
img = cv2.imread(r'panda.jpg',0) #直接读为灰度图像
```

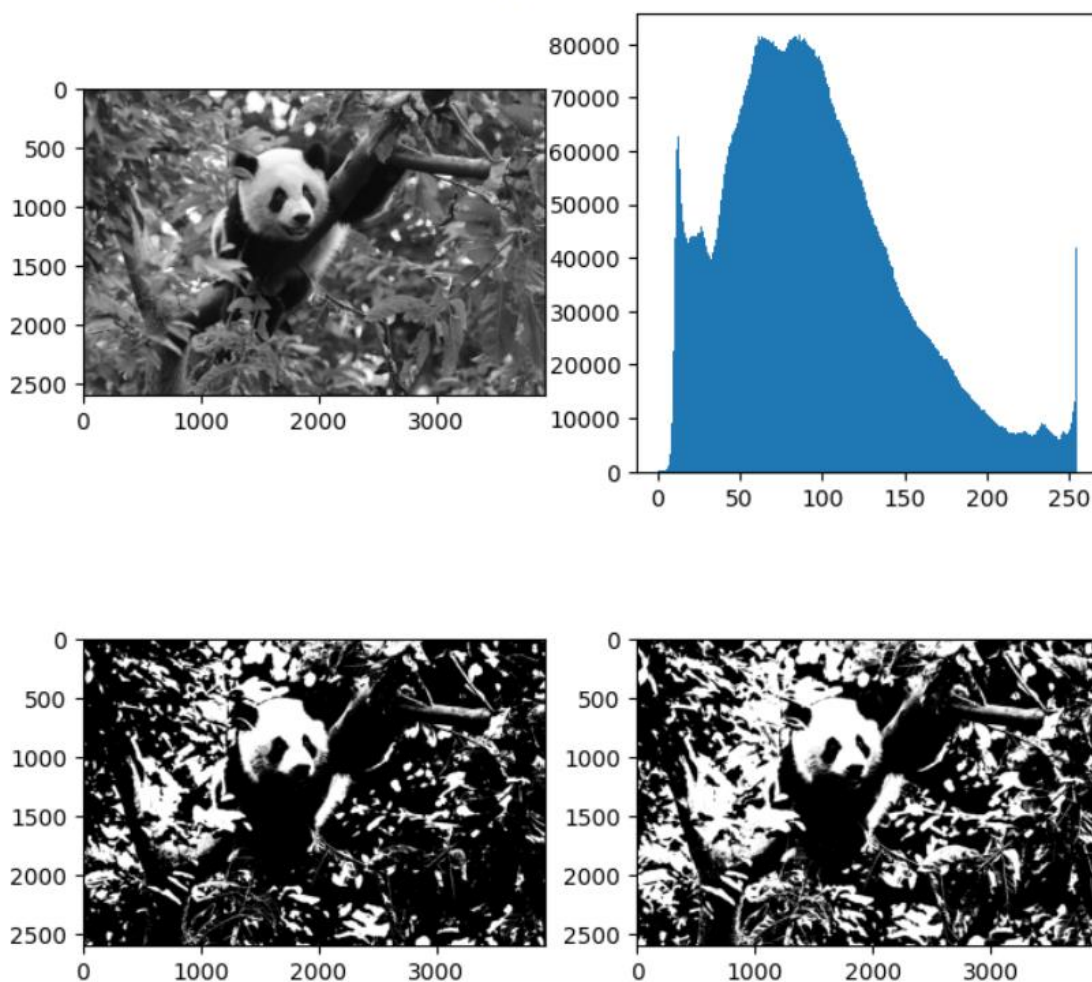
```
#简单分割阈值
ret1,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
#Otsu 滤波
ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_OTSU)
print('简单分割阈值:',ret1)
print('最佳分割阈值:',ret2)
plt.figure(figsize=(8,8))
plt.subplot(221),plt.imshow(img,'gray')
plt.subplot(222),plt.hist(img.ravel(),256)#.ravel 方法将矩阵转化为一维
plt.subplot(223),plt.imshow(th1,'gray')
plt.subplot(224),plt.imshow(th2,'gray')
```

结果：

简单分割阈值: 127.0

最佳分割阈值: 108.0

(<AxesSubplot:~, <matplotlib.image.AxesImage at 0x7fe8241c2b10>)



实验比较了大津阈值法（OTSU）与一般手动阈值法，呈现的图像分割效果。

1.4.5 图像形态学处理

用形态学方法处理二值化图像。

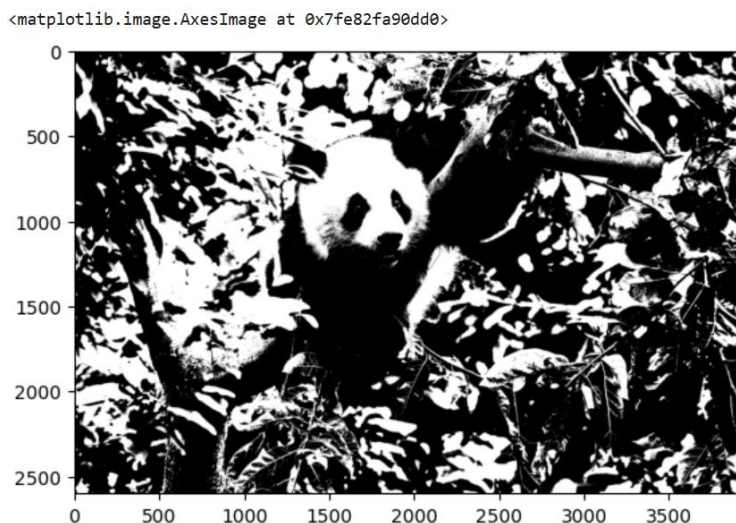
1. 提取的前景目标我们一般设置为白色，背景为黑色。否则，可以使用 `cv2.bitwise_not()` 函数对二值图像反色。
2. 需要先观察图像的目标与噪声。选择合适的形态学操作的组合。“膨胀”就是对图像中的高亮部分进行扩张，让白色区域变多；“腐蚀”就是图像中的高亮部分被蚕食，让黑色区域变多。

步骤 1 读取图像并将图像二值化

代码：

```
img = cv2.imread('panda.jpg',0)
_bin_img = cv2.threshold(img,0,255,cv2.THRESH_OTSU)
plt.imshow(bin_img,'gray')
```

结果：

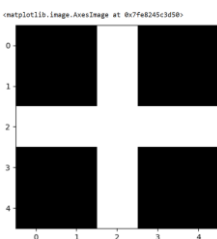


步骤 2 定义结构元素

代码：

```
#定义结构元素
element = cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
plt.imshow(element,'gray')
```

结果：



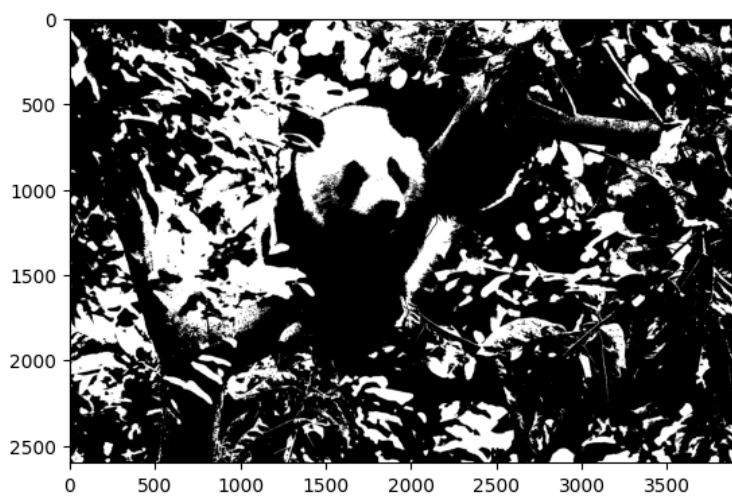
步骤 3 腐蚀

代码:

```
#腐蚀图像
eroded = cv2.erode(bin_img,element)
#显示腐蚀后的图像
plt.imshow(eroded,'gray')
#原图像
plt.figure()
plt.imshow(bin_img,'gray')
```

结果:

<matplotlib.image.AxesImage at 0x7fe8245a0910>



步骤 4 膨胀

代码:

```
#膨胀图像
dilated = cv2.dilate(bin_img,element)
#显示膨胀后的图像
```

```
plt.imshow(dilated,'gray');
#原图像
plt.figure()
plt.imshow(bin_img,'gray')
```

结果:

<matplotlib.image.AxesImage at 0x7fe7fc526850>



1.4.6 图像滤波

步骤 1 中值滤波

代码:

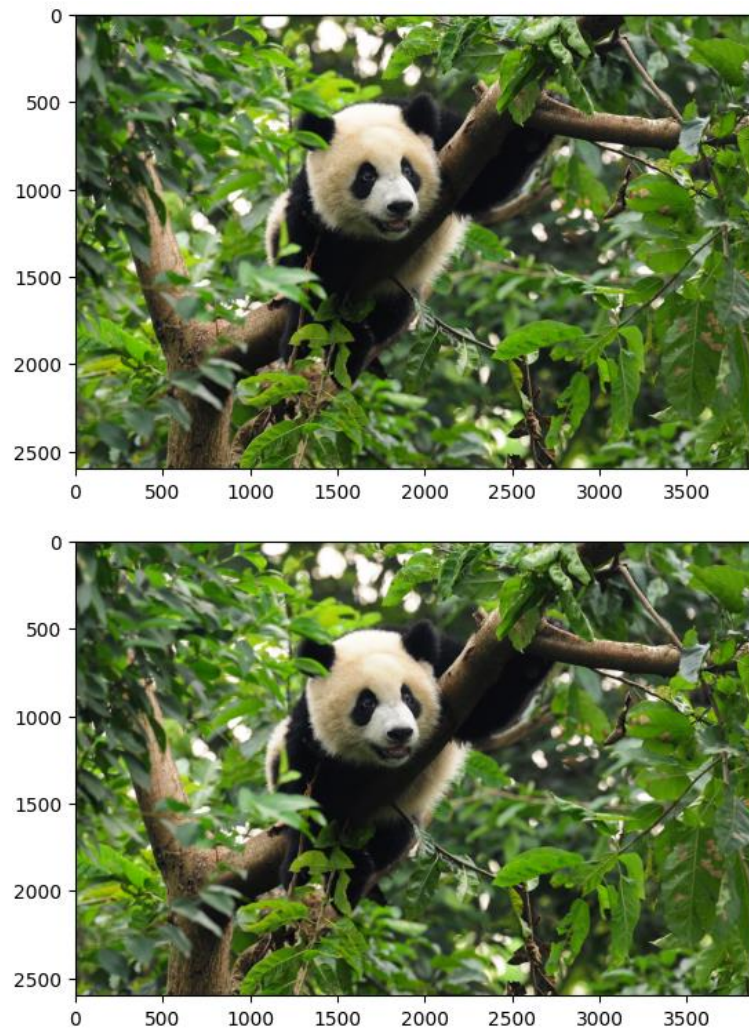
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

im = cv2.imread('panda.jpg')
im = cv2.cvtColor(im,cv2.COLOR_BGR2RGB)
plt.imshow(im)
```

```
# 调用 OpenCV 的中值模糊 API
im_medianblur = cv2.medianBlur(im, 5)

plt.figure()
plt.imshow(im_medianblur)
plt.show()
```

结果：



步骤 2 均值滤波

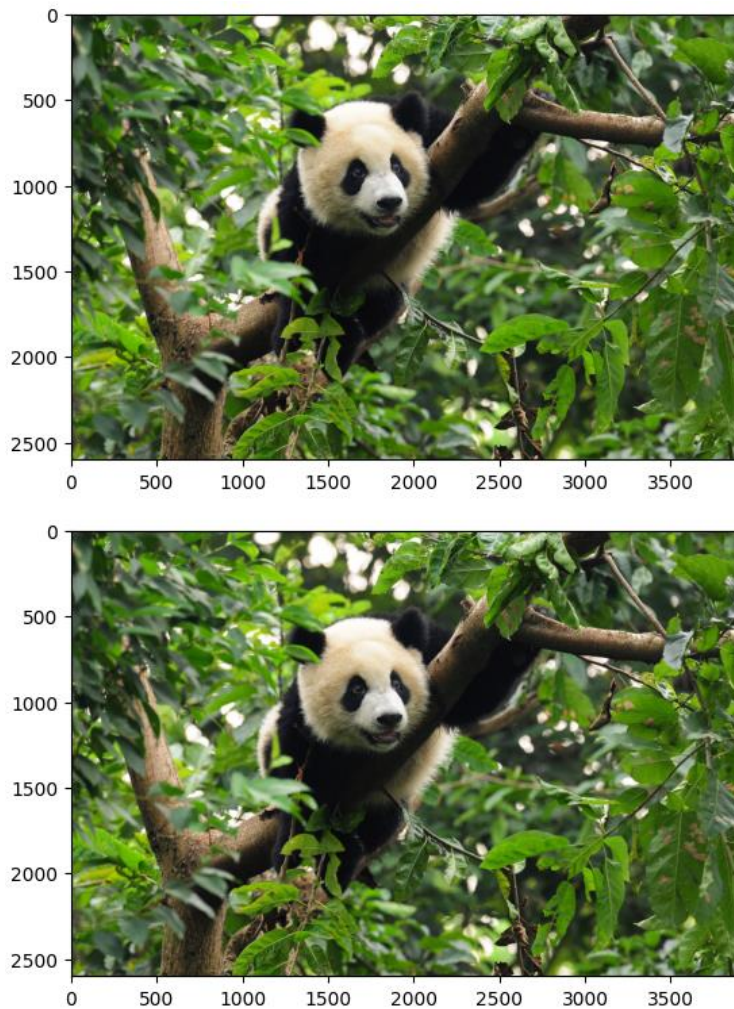
代码：

```
im = cv2.imread('panda.jpg')
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
plt.imshow(im)

# 调用 OpenCV 的均值模糊 API
im_meanblur = cv2.blur(im, (3, 3))
```

```
plt.figure()
plt.imshow(im_meanblur)
plt.show()
```

结果：



步骤 3 高斯滤波

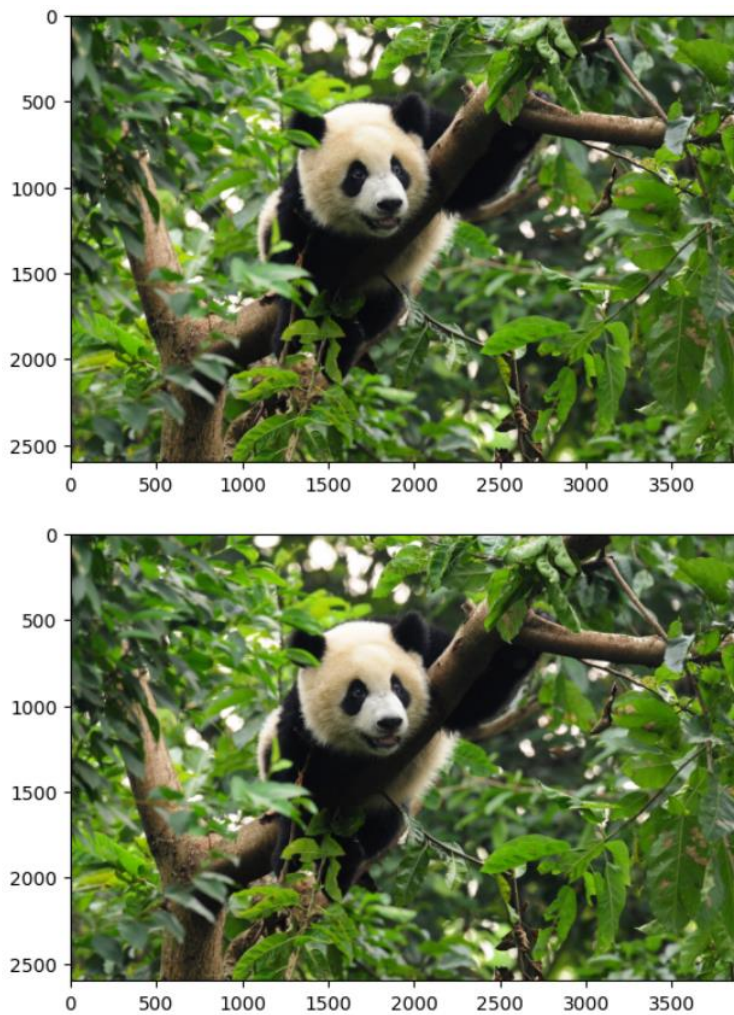
代码：

```
im = cv2.imread('panda.jpg')
im = cv2.cvtColor(im,cv2.COLOR_BGR2RGB)
plt.imshow(im)

# 调用 OpenCV 的高斯模糊 API
im_gaussianblur = cv2.GaussianBlur(im, (5, 5), 0)

plt.figure()
plt.imshow(im_gaussianblur)
plt.show()
```

结果：



步骤 4 图像锐化

代码：

```
im = cv2.imread('panda.jpg')
im = cv2.cvtColor(im,cv2.COLOR_BGR2RGB)
plt.imshow(im)
# 锐化算子
sharpen_1 = np.array([
    [-1,-1,-1],
    [-1,9,-1],
    [-1,-1,-1]])
# 使用 filter2D 进行滤波操作
im_sharpen1 = cv2.filter2D(im,-1,sharpen_1)
plt.figure()
plt.imshow(im_sharpen1)

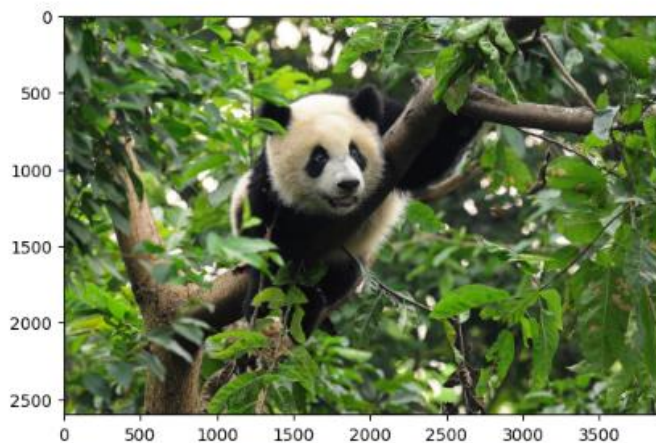
# 锐化算子 2
sharpen_2 = np.array([
    [0,-1,0],
```

```
[-1,8,-1],  
[0,1,0]])/4.0
```

```
# 使用 filter2D 进行滤波操作  
im_sharpen2 = cv2.filter2D(im,-1,sharpen_2)  
plt.figure()  
plt.imshow(im_sharpen2)
```

结果：

<matplotlib.image.AxesImage at 0x7f745027b250>



1.5 实验小结

本节主要介绍了应用基于 Python 语言的 OpenCV 图像处理库进行图像预处理实验的具体操作。使用 OpenCV 图像处理库实现了图像的基本操作、颜色空间转换、几何变换、灰度变换、形态学处理和图像滤波，加深对图像预处理技术的感知，为使用图像预处理技术提供了实际的操作指导。

1.6 思考题

请思考：在图像二值化中，大津法与手动阈值法的区别？

答案：大津阈值法可以自动选择阈值，达到更好的图像分割效果。

2 华为云图像标签实现智慧相册

2.1 实验简介

图像识别是指利用计算机对图像进行处理、分析和理解，以识别各种不同模式的目标和对象的技术。图像识别以开放 API（Application Programming Interface，应用程序编程接口）的方式提供给用户，用户通过实时访问和调用 API 获取推理结果，帮助用户自动采集关键数据，打造智能化业务系统，提升业务效率。

自然图像的语义内容非常丰富，一个图像包含多个标签内容，图像标签可识别三千多种物体以及两万多种场景和概念标签，更智能、准确的理解图像内容，让智能相册管理、照片检索和分类、基于场景内容或者物体的广告推荐等功能更加准确。

信息时代，人们已经习惯了随手拿出手机来拍照，一个人的相册中包含着许多生活的氛围与回忆碎片。然而，信息时代也带来了信息量的爆炸，如果没有经过好好整理，人们的电子设备中或许会存在着数以千计的照片，没有被整理的相片难以串起人们尘封的回忆。

市面上存在着许多制作电子相册的软件，但是多多少少都有自己的限制，同时，有些价格也会比较昂贵。结合华为云服务提供的人工智能 API 与 Python 简单而强大的功能，我们可以自定义我们想要的个性化的相册。

本实验将介绍如何使用华为云图像识别服务，并实现简单的电子相册整理功能。

2.2 实验目的

本实验主要介绍了使用华为云服务中图像识别的功能，通过本实验学员将了解如何结合华为云中的图像识别服务进行图像内容标签的功能。目前华为公有云提供了图像识别的 Restful API 和基于 Python 语言的图像识别的 SDK，本实验将指导学员理解和掌握如何使用 Python 语言利用图像标签业务进行相册的智慧整理。

2.3 实验步骤

该实验需要在华为公有云服务上下载图像识别的 SDK，并通过两种方式进行服务访问，一种是通过 AK\SK 信息进行身份认证从而调用 SDK 底层接口服务进行 Restful 服务请求的提交，另外一种是通过获取用户的 Token 信息模拟浏览器提交 RestFul 请求。具体步骤如下：

2.3.1 申请服务

步骤 1 打开华为云官方网站 <https://www.huaweicloud.com/>



图2-1 华为云官网

步骤 2 使用华为云账号登录之后，点击人工智能产品下的图像识别服务

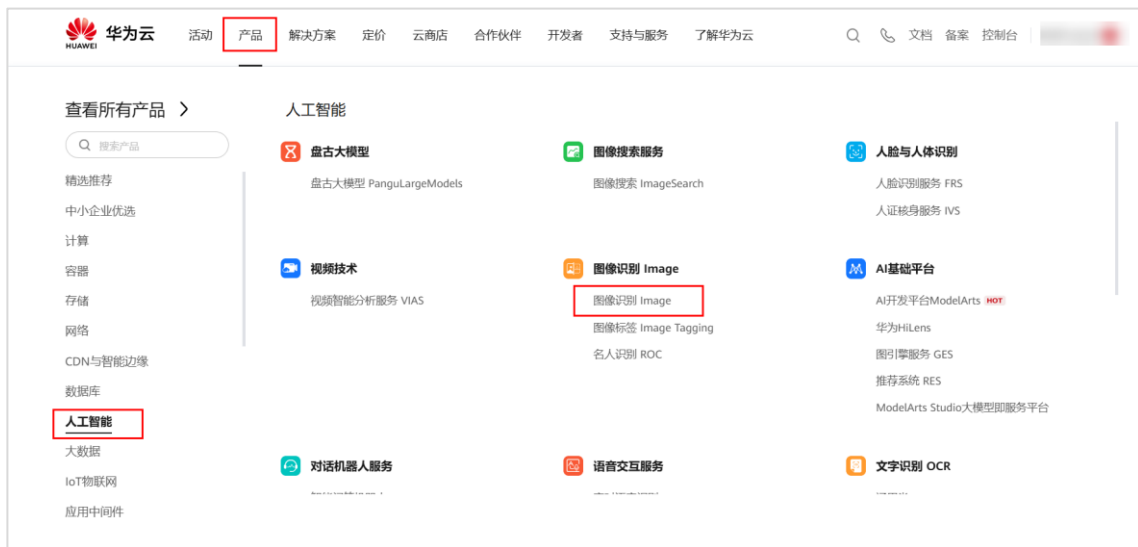


图2-2 图像识别服务

步骤 3 点击控制台



图2-3 服务主界面

步骤 4 选择服务区域‘北京四’，点击服务列表，开通对应的服务，本次实验需要开通图像标签



图2-4 开通服务

2.3.2 通过 AK/SK 进行图像标签服务（如已有，可跳过）

获得 Access Key(AK)与 Secret Access Key(SK)，AK 和 SK 是访问自身账户的密钥，调用图像识别 API 的接口需要用到 AK + SK 密钥。已经获取了 AK + SK 密钥的用户可以跳过此步。

步骤 1 打开华为云官方网站 <https://www.huaweicloud.com/>，进入控制台



图2-5 控制台

步骤 2 点击我的账户下方的我的凭证

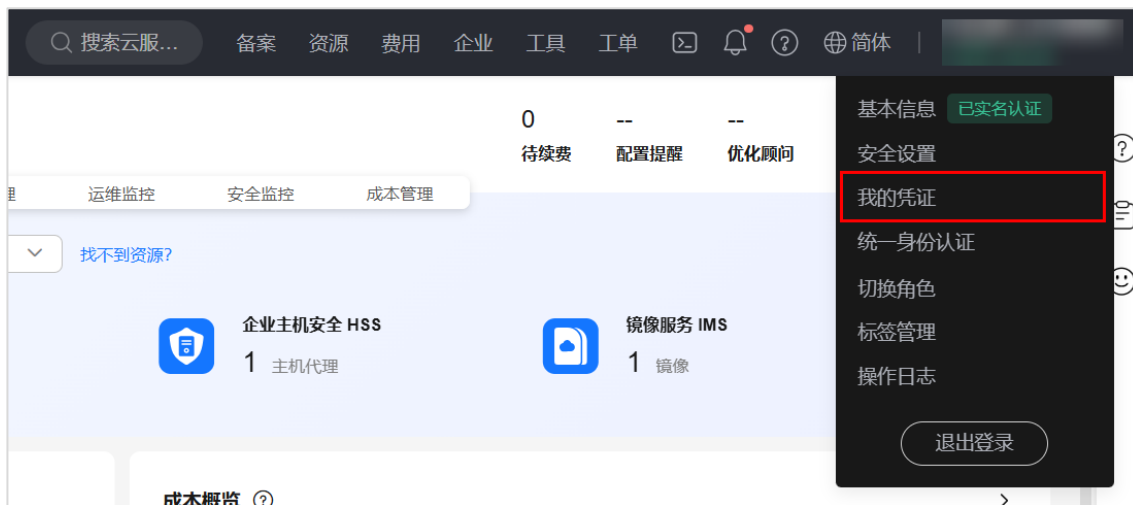


图2-6 我的凭证

步骤 3 在“我的凭证”页面的项目列表中查看项目 ID

项目列表		
项目ID	项目	所属区域
	af-south-1	非洲-约翰内斯堡
	ap-southeast-1	中国-香港
	ap-southeast-3	亚太-新加坡
	cn-east-3	华东-上海一
	cn-north-4	华北-北京四
	cn-north-5	华北-乌兰察布二零一
	cn-north-6	华北-乌兰察布二零二
	cn-north-9	华北-乌兰察布一
	cn-south-1	华南-广州
	cn-southwest-2	西南-贵阳一

图2-7 查看项目 ID

步骤 4 点击访问密钥，新增访问密钥。按照步骤操作以后系统会自动生成下载 csv 文件，密钥就在其中（请妥善保管）



图2-8 下载访问密钥

2.3.3 打开 Jupyter Notebook

可使用本地环境或者华为云 ModelArts（本实验使用华为云 ModelArts 为例）。

2.3.4 安装 Python 版本 SDK

代码：

```
!pip install huaweicloudsdkimage  
!pip install huaweicloudsdkcore
```

2.3.5 图像标签服务初始化

步骤 1 引入相关的库

代码：

```
from huaweicloudsdkcore.auth.credentials import BasicCredentials  
from huaweicloudsdkimage.v2.region.image_region import ImageRegion  
from huaweicloudsdkcore.exceptions import exceptions  
from huaweicloudsdkimage.v2 import *  
  
import matplotlib.pyplot as plt # plt 用于显示图片  
import matplotlib.image as mpimg # mpimg 用于读取图片  
import numpy as np  
  
# 调用 json 解析传回的结果  
import json  
# 操作系统文件/文件夹的包  
import os  
import shutil  
# 图像处理展示相关的包  
from PIL import Image
```

步骤 2 设置相关参数

代码：

```
ak = "请改为自己的 ak"  
sk = "请改为自己的 sk"  
project_id = "请改为自己的 project_id"  
  
credentials = BasicCredentials(ak, sk, project_id)  
  
client = ImageClient.new_builder() \  
    .with_credentials(credentials) \  
    .with_region(ImageRegion.value_of("cn-north-4")) \  
    .build()
```

```
.build()
```

步骤 3 使用本地图像测试

代码：

```
img_tag_path="data/pic2.jpg"
img = mpimg.imread(img_tag_path)
plt.imshow(img) # 显示图片
plt.axis('off') # 不显示坐标轴
plt.show()
```

结果：



代码：

```
import base64
def encode_to_base64(filename):
    """
    encoding file to base64 encoded stream text
    :param filename:
    :return:
    """
    imgstr = ""
    with open(filename, 'rb') as file:
        imgstr = base64.b64encode(file.read())
    return imgstr
request = RunImageTaggingRequest()
request.body = ImageTaggingReq(encode_to_base64(img_tag_path), "", 'en', 80, 5)
result = client.run_image_tagging(request)
print(result)
```

结果：

```
{
  "result": {
    "tags": [
      {
        "confidence": "99.51",
        "i18n_tag": {
          "en": "Cat",
          "zh": "猫"
        },
        "tag": "Cat",
        "type": "object"
      },
      {
        "confidence": "99.03",
        "i18n_tag": {
          "en": "Carnivora",
          "zh": "食肉目"
        },
        "tag": "Carnivora",
        "type": "object"
      },
      {
        "confidence": "88.87",
        "i18n_tag": {
          "en": "Field road",
          "zh": "田野路"
        },
        "tag": "Field road",
        "type": "scene"
      },
      {
        "confidence": "86.07",
        "i18n_tag": {
          "en": "Animal",
          "zh": "动物"
        },
        "tag": "Animal",
        "type": "object"
      },
      {
        "confidence": "83.25",
        "i18n_tag": {
          "en": "Mammal",
          "zh": "哺乳动物"
        },
        "tag": "Mammal",
        "type": "object"
      }
    ]
  }
}
```

2.3.6 利用标签打标相关的照片

步骤 1 打标一张照片

代码：

```
# 确定电子相册位置
file_path = 'data/'
file_name = 'pic8.jpg'

# 保存图片标签的字典
labels = {}

# 图像打标
request = RunImageTaggingRequest()
request.body = ImageTaggingReq(encode_to_base64(file_path + file_name), '', 'zh', 60, 5)
result = client.run_image_tagging(request)
result_dic = json.loads(str(result))
# 保存到字典中
labels[file_name] = result_dic['result']['tags']
print(labels)
```

结果：

```
{'pic8.jpg': [
  {'confidence': '77.85', 'i18n_tag': {'en': 'Dandelion', 'zh': '蒲公英'}, 'tag': '蒲公英', 'type': 'object'},
  {'confidence': '74.6', 'i18n_tag': {'en': 'Plant', 'zh': '植物'}, 'tag': '植物', 'type': 'object'},
  {'confidence': '74.58', 'i18n_tag': {'en': 'Natural', 'zh': '自然'}, 'tag': '自然', 'type': 'object'},
  {'confidence': '72.08', 'i18n_tag': {'en': 'Green', 'zh': '绿色'}, 'tag': '绿色', 'type': 'object'},
  {'confidence': '70.83', 'i18n_tag': {'en': 'Herbal', 'zh': '草本植物'}, 'tag': '草本植物', 'type': 'object'}
]}
```

步骤 2 打标 data 文件夹下的所有照片

代码：

```
# 确定电子相册位置
file_path = 'data/'
# 保存图片标签的字典
labels = {}

items = os.listdir(file_path)
```

```
for i in items:
    # 判断是否为文件，而不是文件夹
    if os.path.isfile:
        # 华为云目前支持 JPG/PNG/BMP 格式的图片
        if i.endswith('jpg') or i.endswith('jpeg') or i.endswith('bmp') or i.endswith('png'):
            # 为图片打上标签
            request = RunImageTaggingRequest()
            request.body = ImageTaggingReq(encode_to_base64(file_path + i), '', 'zh', 60, 5)
            result = client.run_image_tagging(request)
            # 解析返回的结果
            result_dic = json.loads(str(result))
            # 将文件名与图片对齐
            labels[i] = result_dic['result']['tags']

# 显示结果
print(labels)
```

结果：

```
{'pic3.jpg': [{'confidence': '95.41', 'i18n_tag': {'en': 'Lion', 'zh': '狮子'}, 'tag': '狮子', 'type':
'object'}, {'confidence': '91.03', 'i18n_tag': {'en': 'Carnivora', 'zh': '食肉目'}, 'tag': '食肉目',
'type': 'object'}, {'confidence': '87.23', 'i18n_tag': {'en': 'Cat', 'zh': '猫'}, 'tag': '猫', 'type':
'object'}, {'confidence': '86.97', 'i18n_tag': {'en': 'Animal', 'zh': '动物'}, 'tag': '动物', 'type':
'object'}, {'confidence': '74.84', 'i18n_tag': {'en': 'Hairy', 'zh': '毛茸茸'}, 'tag': '毛茸茸', 'type':
'object'}], 'pic8.jpg': [{'confidence': '77.91', 'i18n_tag': {'en': 'Dandelion', 'zh': '蒲公英'}, 'tag':
'蒲公英', 'type': 'object'}, {'confidence': '74.59', 'i18n_tag': {'en': 'Plant', 'zh': '植物'}, 'tag': '植
物', 'type': 'object'}, {'confidence': '74.51', 'i18n_tag': {'en': 'Natural', 'zh': '自然'}, 'tag': '自然
', 'type': 'object'}, {'confidence': '72.19', 'i18n_tag': {'en': 'Green', 'zh': '绿色'}, 'tag': '绿色',
'type': 'object'}, {'confidence': '70.9', 'i18n_tag': {'en': 'Herbal', 'zh': '草本植物'}, 'tag': '草本植
物', 'type': 'object'}], 'pic10.jpg': [{'confidence': '85.83', 'i18n_tag': {'en': 'Flower', 'zh': '花朵
'}, 'tag': '花朵', 'type': 'object'}, {'confidence': '84.33', 'i18n_tag': {'en': 'Plant', 'zh': '植物'},
'tag': '植物', 'type': 'object'}, {'confidence': '83.47', 'i18n_tag': {'en': 'Red', 'zh': '红色'}, 'tag': '
红色', 'type': 'object'}, {'confidence': '79.92', 'i18n_tag': {'en': 'Flower', 'zh': '花'}, 'tag': '花',
'type': 'object'}, {'confidence': '78.67', 'i18n_tag': {'en': 'Flowers and plants', 'zh': '花卉'},
'tag': '花卉', 'type': 'object'}], 'pic_dark.bmp': [{'confidence': '99.22', 'i18n_tag': {'en': 'Lamp',
'zh': '灯'}, 'tag': '灯', 'type': 'object'}, {'confidence': '98.44', 'i18n_tag': {'en': 'Lighting device',
'zh': '照明设施'}, 'tag': '照明设施', 'type': 'object'}, {'confidence': '92.05', 'i18n_tag': {'en': 'Lamp',
'zh': '灯'}, 'tag': '灯', 'type': 'object'}]}
```

步骤 3 保存打标结果

代码：

```
#将标签字典保存到文件
save_path = './label'
# 如果文件夹不存在则创建文件
if not os.path.exists(save_path):
```

```
os.mkdir(save_path)

# 创建文件，执行写入操作，并关闭
with open(save_path + '/labels.json', 'w+') as f:
    f.write(json.dumps(labels))
```

2.3.7 利用打标结果制作动态相册

步骤 1 重新打开保存的标注结果

代码：

```
# 打开刚刚保存的文件
label_path = 'label/labels.json'
with open(label_path, 'r') as f:
    labels = json.load(f)
```

步骤 2 利用关键字进行搜索（这里使用了“花”作为关键字）

代码：

```
# 搜索关键词
key_word = input('请输入搜索词')

# 设置可信百分比
threshold = 60

# 设置一个集合（集合内只存在唯一的元素）
valid_list = set()

# 遍历 labels 中的字典获取所有包含关键字的图片名字
for k, v in labels.items():
    for item in v:
        if key_word in item['tag'] and float(item['confidence']) >= threshold:
            valid_list.add(k)

# 展示结果
valid_list = list(valid_list)
print(valid_list)
```

输入“花”，结果：

请输入搜索词 花

['pic10.jpg', 'pic8.jpg', 'pic7.jpg', 'pic5.jpg', 'pic9.jpg', 'pic4.jpg']

步骤 3 展示相关的图片

代码：

```
# 设置画布大小
plt.figure(24)
```

```
# 将每张图片依次排列到画布上
for k,v in enumerate(valid_list[:9]):
    pic_path = 'data/' + v
    img = Image.open(pic_path)
    img = img.resize((640,400))
    plt.subplot(331 + k)
    plt.axis('off')
    plt.imshow(img)

plt.show()
```

结果：



步骤 4 制作 gif 图片

代码：

```
# 生成一个临时文件夹
if not os.path.exists('tmp'):
    os.mkdir('tmp')

# 将所有搜索到的图像转化为 gif 格式，并存储在临时文件夹中
gif_list = []
for k, pic in enumerate(valid_list):
    pic_path = 'data/' + pic
    img = Image.open(pic_path)
    img = img.resize((640,380))
    save_name = 'tmp/' + str(k) + '.gif'
    img.save(save_name)
    gif_list.append(save_name)

# 打开已经所有静止的 gif 图片
images=[]
for i in gif_list:
```

```

    pic_path = i
    images.append(Image.open(pic_path))

# 存储成动图 gif
images[0].save('相册动图.gif',
               save_all=True,
               append_images=images[1:],
               duration=1000,
               loop=0)

# 释放内存
del images
# 删除临时文件夹
shutil.rmtree('tmp')

print('gif 相册制作完成')

```

结果：

gif 相册制作完成

2.3.8 利用标签对照片进行自动分类

代码：

```

# 打开保存的 labels 文件
label_path = 'label/labels.json'
with open(label_path, 'r') as f:
    labels = json.load(f)
# print(labels)

# 获取置信度最高的文件分类
classes = [[v[0]['tag'], k] for k, v in labels.items()]
print(classes)

for cls in classes:
    if not os.path.exists('data/' + cls[0]):
        os.mkdir('data/' + cls[0])
    # 复制被对应的图片
    shutil.copy('data/' + cls[1], 'data/' + cls[0] + '/' + cls[1])

print('已完成复制!')

```

结果：

```

[['灯', 'pic_dark.bmp'], ['复古', 'pic4.jpg'], ['雏菊', 'pic5.jpg'], ['人', 'pic_super.png'], ['狮子', 'pic3.jpg'], ['蒲公英', 'pic7.jpg'], ['奔跑', 'pic1.jpg'], ['花朵', 'pic10.jpg'], ['玫瑰', 'pic9.jpg'], ['猫', 'pic2.jpg'], ['蒲公英', 'pic8.jpg'], ['房子', 'pic_fog.png'], ['狗', 'pic6.jpg']]

```

已完成复制!

2.4 实验小结

本次实验介绍如何利用华为云图像标签服务进行了电子相册的相关功能操作，实验首先介绍了如何开通华为云图像识别下面的服务项目，重点介绍如利用图像标签服务完成相片的标注，相册的搜索，动态相册制作，照片的自动分类等方法，并且展示相关结果。同时，对常见的华为云图像识别的库的基本操作进行了实践和操作。

2.5 思考题

请思考：如何在 Notebook 中展示 gif 动图？

答案：运行以下代码

```
from IPython.display import Image  
Image(url='./相册动图.gif')
```

3 使用卷积神经网络实现图像分类

3.1 实验简介

在学习了图像卷积的基本操作与卷积神经网络的工作方式后，为加强学员对卷积神经网络的理解，本实验将利用 pytorch 2.5.1 实现基于 Cifar10 的彩色图像分类。

3.2 实验目的

实验通过代码讲解的方式，帮助大家掌握神经网络构建的全过程。实验步骤包括：数据预处理、模型创建、模型编译、模型训练和模型测试。

3.3 实验步骤

3.3.1 数据预处理

步骤 1 导入工具包

检查 pytorch 版本代码：

```
import torch
# 检查 PyTorch 版本
print(torch.__version__)
```

输出：

```
2.5.1
```

导入工具包：

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
import os
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader
```

步骤 2 数据集预处理

代码：

```
# 下载 CIFAR10 数据集
transform = transforms.Compose([
    transforms.ToTensor(), # 将图片转换为 Tensor
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) # 归一化
])

train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

# 创建 DataLoader
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

结果：

```
#首次运行此代码会输出如下结果，并且在 modelarts 文件侧会生成用于放数据的 data 文件夹：
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170M/170M [00:13<00:00, 12.9MB/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified

#再次运行此代码会输出如下结果，表示数据已经下载过：
Files already downloaded and verified
Files already downloaded and verified
```

代码：

```
# 打印数据集图像的大小
print(train_dataset.data.shape, test_dataset.data.shape)
# 将分类标签变为 onehot 编码
num_classes = 10
y_train_onehot = torch.nn.functional.one_hot(torch.tensor(train_dataset.targets), num_classes)
y_test_onehot = torch.nn.functional.one_hot(torch.tensor(test_dataset.targets), num_classes)
print(train_dataset.targets[0])
print(y_train_onehot[0])
```

结果：

```
(50000, 32, 32, 3) (10000, 32, 32, 3)
6
tensor([0, 0, 0, 0, 0, 0, 1, 0, 0, 0])
```

步骤 3 绘制前 9 张图像

代码：

```
# 创建图像标签列表
```

```
category_dict = {0:'airplane',1:'automobile',2:'bird',3:'cat',4:'deer',5:'dog',
                 6:'frog',7:'horse',8:'ship',9:'truck'}

# 绘制前 9 张训练图像，并输出图像标签
# 新建一张空白绘图区域
plt.figure(figsize=(5, 5)) # 可以调整图像大小
for i in range(9):
    # 将绘图区域分成 9 个小区域
    plt.subplot(3, 3, i+1)

    # 获取图像和标签
    img, label = train_dataset[i]

    # 反归一化，以便正确显示图像
    img = img / 2 + 0.5 # 反归一化
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0))) # 转换为 numpy 数组并调整维度

    # 打印图像标签
    plt.title(category_dict[label])
    plt.axis('off') # 不显示坐标轴

plt.show()
```

结果：



3.3.2 构建卷积神经网络

利用 pytorch 构建卷积神经网络模型。步骤包括定义模型结构、实例化优化器，编译模型。整个网络包括了卷积层、池化层、dropout 层和全连接层。

步骤 1 定义模型结构、编译模型

代码：

```
#输入图像大小为 ( 32, 32, 3 )
class CNNClassificationModel(nn.Module):
    def __init__(self, num_classes=10):
        super(CNNClassificationModel, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)
        self.dropout1 = nn.Dropout(p=0.25)

        self.conv3 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)
        self.dropout2 = nn.Dropout(p=0.25)

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(in_features=64 * 8 * 8, out_features=512)
        self.dropout3 = nn.Dropout(p=0.5)
        self.fc2 = nn.Linear(in_features=512, out_features=num_classes)

        self.MaxPool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.ReLU = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.conv1(x)
        x = self.ReLU(x)
        x = self.conv2(x)
        x = self.ReLU(x)
        x = self.MaxPool2d(x)
        x = self.dropout1(x)

        x = self.conv3(x)
        x = self.ReLU(x)
        x = self.conv4(x)
        x = self.ReLU(x)
        x = self.MaxPool2d(x)
        x = self.dropout2(x)

        x = self.flatten(x)
        x = self.fc1(x)
        x = self.ReLU(x)
```

```
x = self.dropout3(x)
x = self.fc2(x)
x = self.softmax(x)
return x
```

步骤 2 模型实例化

代码：

```
#网络实例化
# 定义超参数
num_classes = 10
learning_rate = 0.0001
weight_decay = 1e-6
#判断设备是否有 cuda
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Using device: {device}')
# 实例化模型且将模型放到 GPU 上运行
model = CNNClassificationModel(num_classes=num_classes).to(device)

# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer = optim.RMSprop(model.parameters(), lr=learning_rate, weight_decay=weight_decay)

# 打印模型结构
print(model)
```

结果：

```
Using device: cuda
CNNClassificationModel(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dropout1): Dropout(p=0.25, inplace=False)
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dropout2): Dropout(p=0.25, inplace=False)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=4096, out_features=512, bias=True)
  (dropout3): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=512, out_features=10, bias=True)
  (MaxPool2d): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (ReLU): ReLU()
  (softmax): Softmax(dim=1)
)
```

3.3.3 模型训练

在训练集上训练模型，并利用 checkpoint 回调函数保存训练过程中精确度最高的模型参数。

代码：

```
#在北京四以外的其他区域的 notebook 执行上述代码出现数据下载超时，添加下方网络环境配置。
%env no_proxy=repo.myhuaweicloud.com,modelarts.cn-north-9.myhuaweicloud.com,obs.cn-north-9.myhuaweicloud.com,iam.cn-north-9.huaweicloud.com,iam-cache-proxy.cn-north-9.myhuaweicloud.com,dli.cn-north-9.myhuaweicloud.com,127.0.0.1,localhost,172.16.*
%env NO_PROXY=repo.myhuaweicloud.com,modelarts.cn-north-9.myhuaweicloud.com,obs.cn-north-9.myhuaweicloud.com,iam.cn-north-9.huaweicloud.com,iam-cache-proxy.cn-north-9.myhuaweicloud.com,dli.cn-north-9.myhuaweicloud.com,127.0.0.1,localhost,172.16.*
#下载预训练权重
!wget https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIP-AI%20EI%20Developer/V2.5/cifar10_weights.pth
#下载成功后会出现 cifar10_weights.pth 权重文件。
```

代码：

```
#此段用于加载预训练模型权重，加载预训练模型后，下方训练精度提升程度不大。
#如果需要不加载预训练模型，不运行此段，直接运行下方代码可看见清晰的 loss 下降和精度提升

if os.path.exists('cifar10_weights.pth'):

    model.load_state_dict(torch.load('cifar10_weights.pth', map_location=device, weights_only=True))
    print("Pre-trained weights loaded successfully.")
else:
    print("Pre-trained weights file not found. Training from scratch.")
```

结果：

```
#输出如下结果表示预训练权重加载成功
Pre-trained weights loaded successfully.
```

代码：

```
num_epochs=10
best_loss = float('inf')

for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0
    for i, (images, labels) in enumerate(train_loader):
        # 将数据移动到选定的设备上
        images = images.to(device)
        labels = labels.to(device)

        # 前向传播
        outputs = model(images)
        loss = criterion(outputs, labels)
```

```
# 反向传播和优化
optimizer.zero_grad()
loss.backward()
optimizer.step()

total_loss += loss.item()

# 每 200 个步骤输出一一次训练损失
if (i + 1) % 200 == 0:
    print(f'Epoch [{epoch + 1}/{num_epochs}], Step [{i + 1}/{len(train_loader)}], Train Loss: {loss.item():.4f}')

# 计算平均训练损失
avg_train_loss = total_loss / len(train_loader)

# 验证模型
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        # 将数据移动到选定的设备上
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

# 计算验证精度
accuracy = 100 * correct / total

# 输出训练损失和验证精度
print(f'Epoch [{epoch + 1}/{num_epochs}], Train Loss: {avg_train_loss:.4f}, Val Accuracy: {accuracy:.2f}%')

# 保存最佳模型参数
if avg_train_loss < best_loss:
    best_loss = avg_train_loss
    torch.save(model.state_dict(), 'final_cifar10.pth')
    print(f'Saved model with loss: {best_loss:.4f}')
```

结果:

```
Epoch [1/10], Step [200/782], Train Loss: 1.6236
Epoch [1/10], Step [400/782], Train Loss: 1.6530
Epoch [1/10], Step [600/782], Train Loss: 1.6521
```

```
Epoch [1/10], Train Loss: 1.6474, Val Accuracy: 74.06%
Saved model with loss: 1.6474
Epoch [2/10], Step [200/782], Train Loss: 1.6274
Epoch [2/10], Step [400/782], Train Loss: 1.6542
Epoch [2/10], Step [600/782], Train Loss: 1.6024
Epoch [2/10], Train Loss: 1.6420, Val Accuracy: 73.85%
.....
.....
Epoch [10/10], Step [600/782], Train Loss: 1.5505
Epoch [10/10], Train Loss: 1.6275, Val Accuracy: 74.85%
Saved model with loss: 1.6275
```

结果说明：加载了预训练模型后，随着 epoch 数量的增加，模型精确度增大。当 epochs=10 时，模型精确度为 74.85%。还需要继续增大 epoch 的数量，提升模型精确度。

3.3.4 测试网络模型

在测试集上，测试网络模型的精确度。

步骤 1 测试集上测试模型

代码：

```
new_model = CNNClassificationModel(num_classes=10).to(device)
new_model.load_state_dict(torch.load('final_cifar10.pth', map_location=device, weights_only=True))
new_model.eval()

# 生成预测结果
predicted = []
y_test = []
with torch.no_grad():
    for i in range(4):
        img, label = test_dataset[i]
        img = img.unsqueeze(0).to(device) # 增加一个批次维度
        label = torch.tensor(label).to(device)

        # 前向传播
        output = new_model(img)
        _, pred = torch.max(output.data, 1)

        # 存储预测结果和实际标签
        predicted.append(pred.item())
        y_test.append(label.item())

# 打印预测结果
for i in range(4):
    print(f'Image {i+1}:')
    print(f' Ground Truth: {category_dict[y_test[i]]}')
```

```
print(f' Predicted: {category_dict[predicted[i]]}')
```

结果：

```
Image 1:
  Ground Truth: cat
  Predicted: cat
Image 2:
  Ground Truth: ship
  Predicted: ship
Image 3:
  Ground Truth: ship
  Predicted: ship
Image 4:
  Ground Truth: airplane
  Predicted: airplane
```

步骤 2 显示测试集中前 4 张图像的预测输出

代码：

```
#在图像标签列表中找到对应的预测标签
#输出可视化结果
plt.figure(figsize=(8, 8)) # 可以调整图像大小
for i in range(4):
    plt.subplot(2, 2, i + 1)
    img, label = test_dataset[i]
    img = img.cpu().numpy().transpose((1, 2, 0))
    img = img * np.array([0.5, 0.5, 0.5]) + np.array([0.5, 0.5, 0.5]) # 反归一化
    plt.imshow(img)
    plt.title(f'pred: {category_dict[predicted[i]]}   actual: {category_dict[y_test[i]]}')
    plt.axis('off')
plt.show()
```

结果：



3.4 实验小结

本实验讲述了使用卷积神经网络实现彩色图像分类应用，加深了学员对深度学习的理解：通过该实验，学员能更好地理解卷积神经网络提取图像的过程；增强对卷积神经网络中卷积层、池化层和全连接层的理解和使用。

3.5 思考题

请思考：神经网络模型实现的流程包括哪些步骤？

答案：模型构建、编译、训练、测试、保存和加载。

华为认证 AI 系列教程

HCIP-AI-EI Developer

华为企业AI开发高级工程师认证

语音处理

实验指导手册

版本:2.5



华为技术有限公司

版权所有 © 华为技术有限公司 2026。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<https://e.huawei.com>

华为认证体系介绍

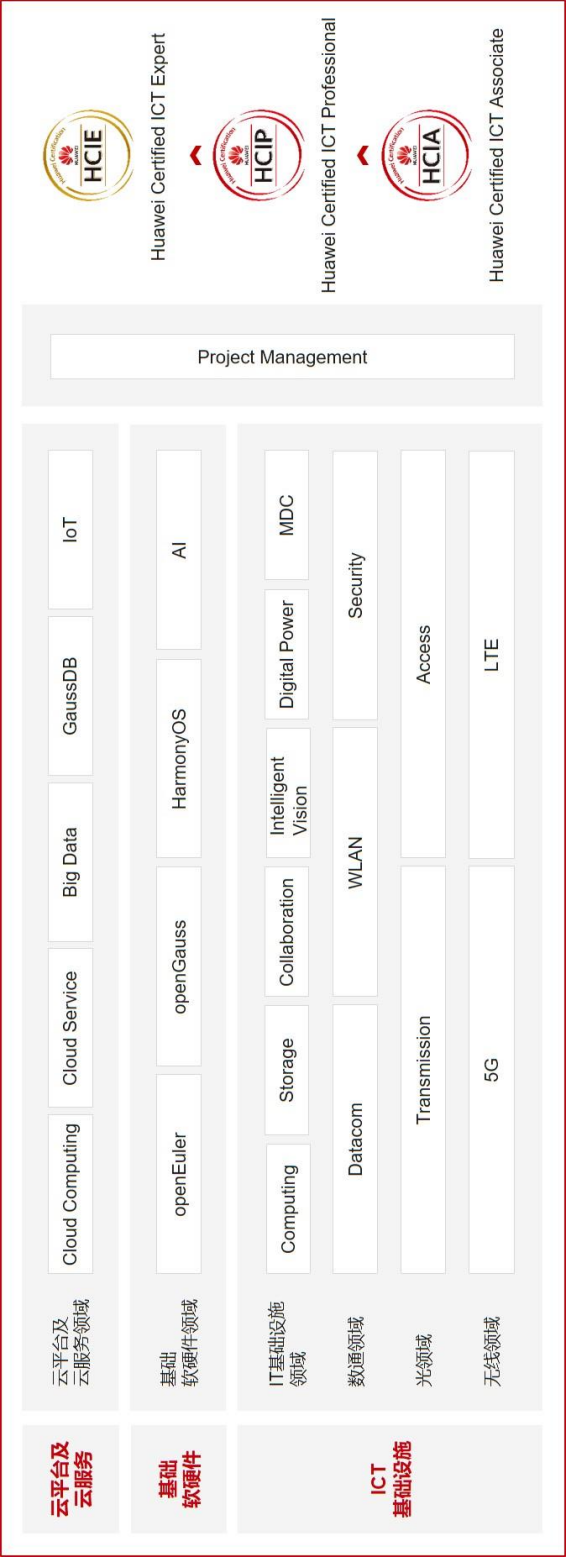
华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT基础设施认证、基础软硬件认证、云平台及云服务认证三类认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIP-AI-EI Developer V2.5定位于培养与认证具有图像处理、语音处理、自然语言处理能力，能够应用通用开源框架和华为云ModelArts平台等进行开发和创新的高级工程师。

通过HCIP-AI-EI Developer V2.5认证，您将掌握神经网络基础、图像处理理论、语音处理理论、自然语言处理理论、大模型深度学习算法等基础知识，具备运用通用开源框架和华为云ModelArts平台等进行图像处理、语音处理和自然语言处理等人工智能产品设计开发的能力，能够胜任AI开发高级工程师、图像处理开发工程师、语音处理开发工程师、自然语言处理开发工程师等岗位。



简介

本书为 HCIP-AI-EI Developer V2.5 认证培训教程，适用于准备参加 HCIP-AI-EI Developer V2.5 考试的学员或者希望了解 AI 基础知识的读者。掌握本实验手册内容，您将能够了解如何进行语音文件预处理、语音输入、语音合成和语音识别等相关语音处理的实际开发。而对于语音识别和语音合成的操作主要是通过提交服务请求调用华为公有云服务中的语音识别和语音合成服务，应用华为公有云底层的语音服务进行语音的相关处理。掌握包括语音文件预处理，语音合成以及短语音识别的操作。

通过深度学习的语音识别的操作，掌握语音识别的数据处理和深度学习进行语音识别的流程。

内容描述

本实验指导书共包含 3 组实验，涉及到基于 Python 的传统语音识别，语音文件预处理，和基于 Python 的语音合成和语音识别的内容，以及通过 Seq2Seq 进行语音识别的内容。提升 AI 语音处理的实践开发能力。

实验 1 了解 Python 语音文件预处理。

实验 2 掌握如何调用华为云的语音识别和语音合成的接口。

实验 3 掌握如何通过 Seq2Seq 进行语音识别。

读者知识背景

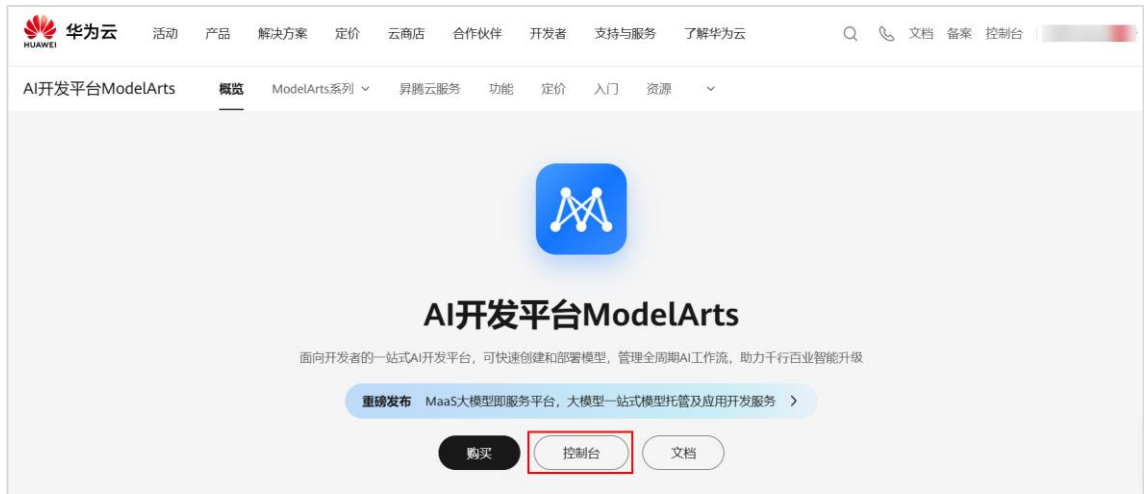
本课程为华为认证开发课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 有一定的 Python 语言编程基础和语音数据预处理基础。
- 有一定的语音识别和语音合成相关的理论基础。
- 有一定的深度学习 Seq2Seq 的理论基础。

准备实验环境

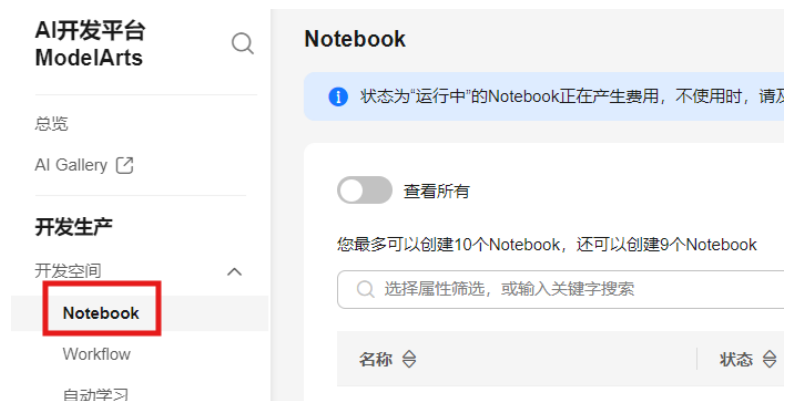
步骤 1 进入华为云 ModelArts 控制台

在[华为云 ModelArts 主页](https://www.huaweicloud.com/product/modelarts.html)（<https://www.huaweicloud.com/product/modelarts.html>），点击“控制台”进入 ModelArts 的管理页面。



步骤 2 创建 Notebook 训练作业

在左侧菜单栏中选择“开发空间”的“Notebook”，点击进入 Notebook 创建页面。



点击“创建 Notebook”按钮，创建一个新的 Notebook，其配置如下：

- 名称：自定义
- 自动停止：建议 2 小时
- 镜像：pytorch2.5_cuda11.8
- 资源池：公共资源池
- 类型：CPU/GPU
- 规格：根据需求选取（实验中数据量较小，可选取较低配置）
- 存储配置：云硬盘 EVS
- 磁盘规格：5GB

如图所示：

* 名称

notebook-hcip-image

描述

0/512

* 自动停止

开启该选项后，该Notebook实例将在运行时长超出您所选择的时长后，自动停止。

X

☐ 1 小时

☒ 2 小时

☐ 4 小时

☐ 6 小时

☐ 自定义

* 镜像

公共镜像

自定义镜像 ?

请输入镜像名称

Q

Q

名称	版本	所属组织	描述
☐ pytorch2.5_cuda11.8	v1	ai-certification	pytorch2.5.1+GPU
☒ pytorch2.5_cuda11.8	1.1	ai-certification	--

* 资源类型

公共资源池

专属资源池

* 类型

GPU

* 实例规格

GPU: 1*Vnt1(16GB)|CPU: 8vCPUs 64GB

▼

One GP Vnt1 GPU with 16GB of memory, ideal for deep learning algorithm training and debugging.

* 存储配置

云硬盘EVS ?

云硬盘EVS作为持久化存储挂载在/home/ma-user/work目录下，该目录下的内容在实例停止后会被保留，存储支持在线按需扩容

* 磁盘规格

-

20

+

GB

磁盘规格默认为5GB，从Notebook实例创建成功起，直至删除成功，每GB按照规定费用收费。

SSH远程开发

★ 镜像

公共镜像

自定义镜像

?

请输入镜像名称

Q

Q

	名称	版本	所属组织	描述
☐	pytorch2.5_cuda11.8	v1	ai-certification	pytorch2.5.1+GPU
☒	pytorch2.5_cuda11.8	1.1	ai-certification	--

★ 资源类型

公共资源池

专属资源池

★ 类型

GPU

★ 实例规格

GPU: 1*Vmt1(16GB)CPU: 8vCPUs 64GB

One GP Vmt1 GPU with 16GB of memory, ideal for deep learning algorithm training and debugging.

★ 存储配置

云硬盘EVS

?

云硬盘EVS作为持久化存储挂载在/home/ima-user/work目录下, 该目录下的内容在实例停止后会被保留, 存储支持在线按需扩容

★ 磁盘规格

-

20

+

GB

磁盘规格默认为5GB, 从Notebook实例创建成功起, 直至删除成功, 每GB按照规定费用收费。

SSH远程开发

☐

配置完成之后“立即创建”，规格确认无误之后“提交”。

步骤 3 启动 Notebook 进入开发环境

当上一步创建好 Notebook 状态显示为“运行中”时，在右侧“操作”栏中“打开”，即可进入在线编程页面。

目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
准备实验环境	3
1 语音预处理	8
1.1 实验介绍	8
1.1.1 关于本实验	8
1.1.2 实验目的	8
1.1.3 实验预备知识	8
1.2 实验步骤	9
1.3 思考题	18
1.4 实验小结	18
2 语音合成和语音识别	19
2.1 实验介绍	19
2.1.1 关于本实验	19
2.1.2 实验目的	19
2.1.3 实验环境准备	19
2.1.4 SDK 安装	20
2.2 实验步骤	23
2.2.1 语音合成	23
2.2.2 语音识别	25
2.3 实验小结	27
3 基于 Seq2Seq 的语音识别	28
3.1 实验介绍	28
3.1.1 关于本实验	28
3.1.2 实验目的	28
3.1.3 实验预备知识	28
3.2 实验步骤	28
3.3 思考题	36



3.4 实验小结	36
----------------	----

1 语音预处理

1.1 实验介绍

1.1.1 关于本实验

语音是一种非平稳的时变信号，它携带着各种信息。在语音编码、语音合成、语音识别和语音增强等语音处理中都需要提取语音中包含的各种信息。一般而言，语音数据处理的目的有两种：一种是对语音信号进行分析，提取特征参数，用于后续处理；另一种是加工语音信号，例如在语音增强中对含噪语音进行背景噪声抑制，以获得相对“干净”的语音；在语音合成方法中需要对分段语音进行拼接平滑，获得主观音质较高的合成语音，这方面的应用同样是建立在分析并提取语音信号信息的基础上的。总之，语音信号分析的目的就在于方便有效地提取并表示语音信号所携带的信息。

根据所分析的参数类型，语音信号分析可以分成时域分析和变换域（频域、倒谱域）分析。其中时域分析方法是最简单、最直观的方法，它直接对语音信号的时域波形进行分析，提取的特征参数主要有语音的短时能量和平均幅度、短时平均过零率、短时自相关函数和短时平均幅度差函数等。

本实验就是基于开源的短序列语音数据集对语音数据本身的属性，以及相关特征属性进行分析与解析，以求对语音数据有更加深入全面的了解。

1.1.2 实验目的

- 查看语音数据的属性。
- 了解语音数据的特征。

1.1.3 实验预备知识

本实验操作需要具备 2 方面的预备知识：

- 有相应 Python 语言语法基础，能动手实操。
 - 对相关框架 wave 有一定了解。
-

1.2 实验步骤

本章内容主要是基于 wave 进行操作。主要步骤包括：

- 音频数据属性查看
- 音频数据转换矩阵
- 音频频谱
- 音频波形

步骤 1 下载数据集

获取数据文件：

```
!wget https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIP-AI%20EI%20Developer/V2.5/chapter3/speech.zip
# 解压文件
!unzip speech.zip
```

步骤 2 导入相关模块

代码如下：

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

步骤 3 查看 wav 文件基本属性

需要先下载数据，解压后找到“实验 1-语音预处理”中的 data 文件夹，放在当前工作目录下。

然后编写如下代码：

```
filename = 'data/thchs30/train/A2_0.wav '

# 输出信息（声道，采样宽度，帧速率，帧数，唯一标识，无损）
WAVE = we.open(filename)

for item in enumerate(WAVE.getparams()):
    print(item)
a = WAVE.getparams().nframes    # 帧总数
```

```
print(a)
f = WAVE.getparams().framerate # 采样频率
print("采样频率: ",f)
sample_time = 1/f             # 采样点的时间间隔
time = a/f                    # 声音信号的长度
sample_frequency, audio_sequence = wavfile.read(filename)
print (audio_sequence,len(audio_sequence)) # 声音信号每一帧的“大小”
x_seq = np.arange(0,time,sample_time)
print(x_seq,len(x_seq))
```

输出结果:

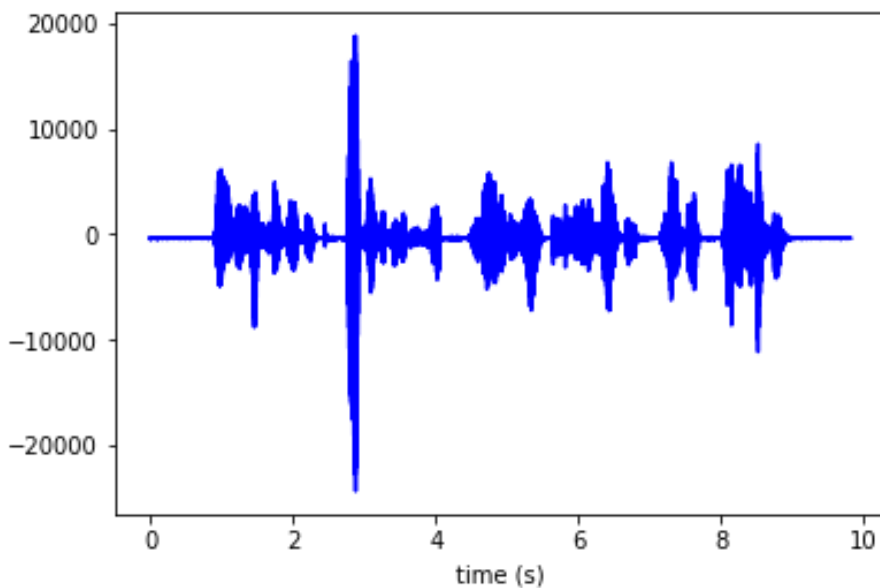
```
(0, 1)
(1, 2)
(2, 16000)
(3, 157000)
(4, 'NONE')
(5, 'not compressed')
157000
采样频率: 16000
[-296 -424 -392 ... -394 -379 -390] 157000
[0.00000000e+00 6.25000000e-05 1.25000000e-04 ... 9.8123125e+00 9.8123750e+00
 9.8124375e+00] 157000
```

步骤 4 查看 wav 文件波形序列

代码如下:

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

输出结果:



步骤 5 文件名获取

代码如下（数据在步骤 2 已经下载）：

```
audio_path = 'data/train/audio/'
pict_Path = 'data/train/audio/'
samples = []
# 验证文件是否存在，若不在这，则创建
if not os.path.exists(pict_Path):
    os.makedirs(pict_Path)

subFolderList = []
for x in os.listdir(audio_path):
    if os.path.isdir(audio_path + '/' + x):
        subFolderList.append(x)
        if not os.path.exists(pict_Path + '/' + x):
            os.makedirs(pict_Path + '/' + x)
# 查看子文件名称和数量
print("----list----:",subFolderList)
print("----len----:",len(subFolderList))
```

输出结果：

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

步骤 6 统计每个子文件夹语音文件数量

代码如下：

```
# count
sample_audio = []
total = 0
for x in subFolderList:
    # 获取所有 wav 文件
    all_files = [y for y in os.listdir(audio_path + x) if '.wav' in y]
    total += len(all_files)
    sample_audio.append(audio_path + x + '/' + all_files[0])
    # print(sample_audio)
    # 查看每个子文件夹文件数量
    print('%s : count: %d ' % (x, len(all_files)))
# 查看 wav 文件总数
print("TOTAL:",total)
```

输出结果:

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

TOTAL: 521

步骤 7 查看每个子文件当中第一个文件

代码如下：

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

输出结果：

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

步骤 8 构建频谱处理函数

代码如下：

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

步骤 9 多样本单个频谱可视化

代码如下：

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

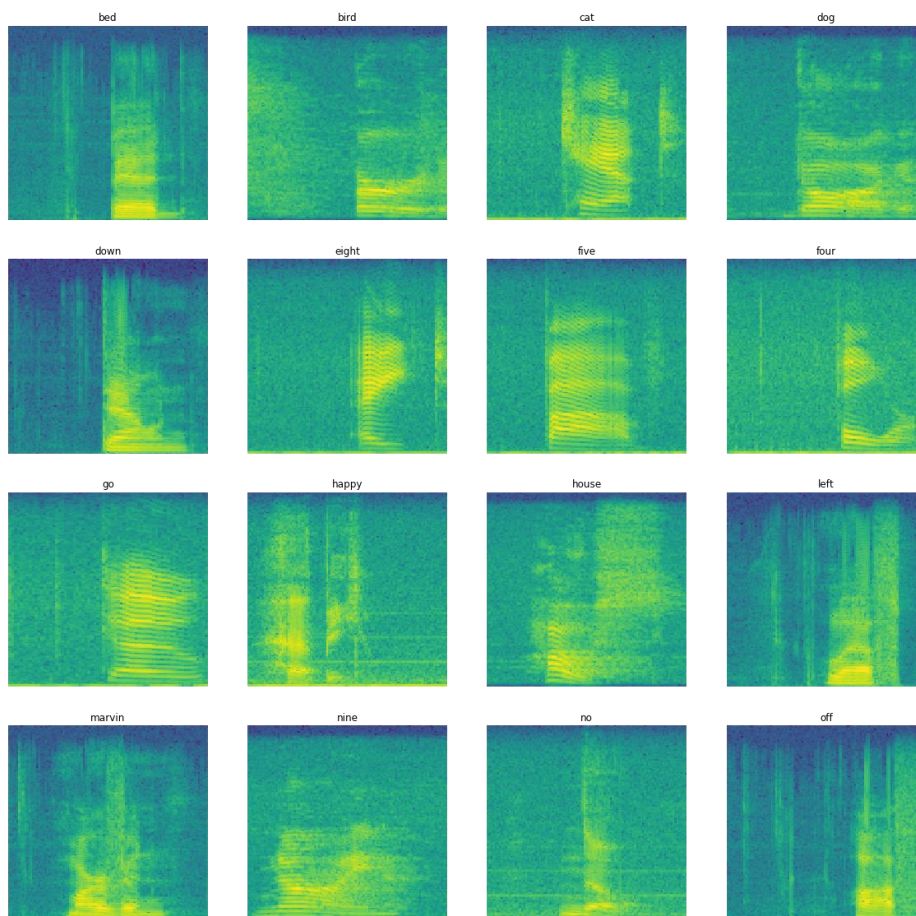
for i, filepath in enumerate(sample_audio[:16]):
    # Make subplots
    plt.subplot(4,4,i+1)

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

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

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

输出结果：



步骤 10 单样本多个频谱可视化

代码如下：

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

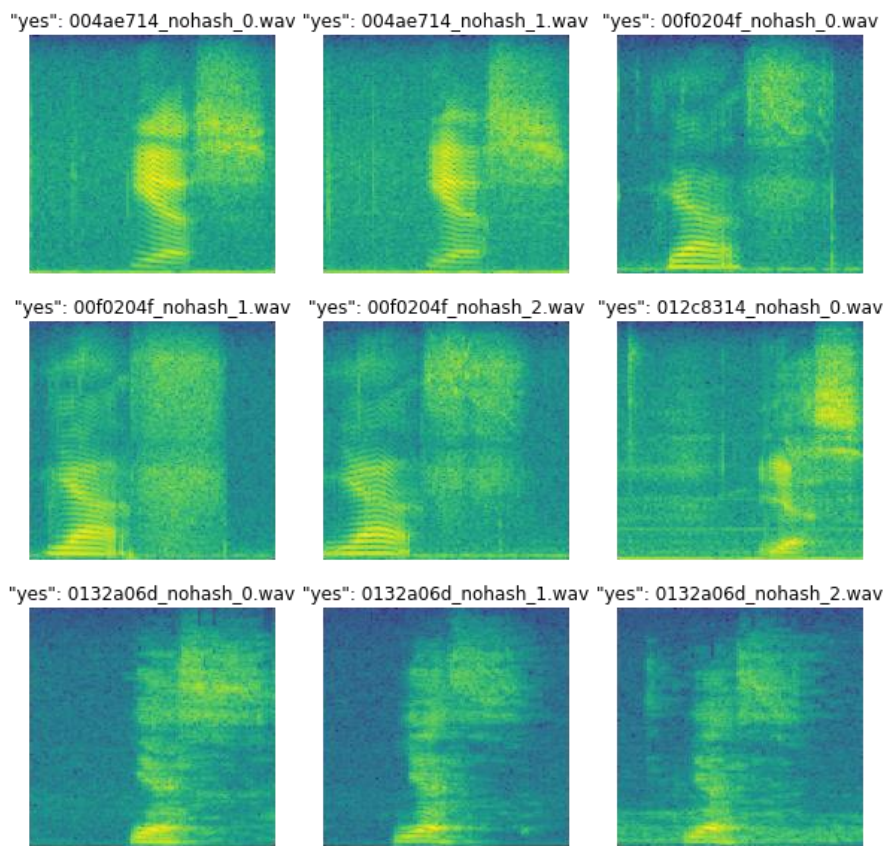
for i, filepath in enumerate(yes_samples):
    # Make subplots
    plt.subplot(3,3,i+1)

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

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

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

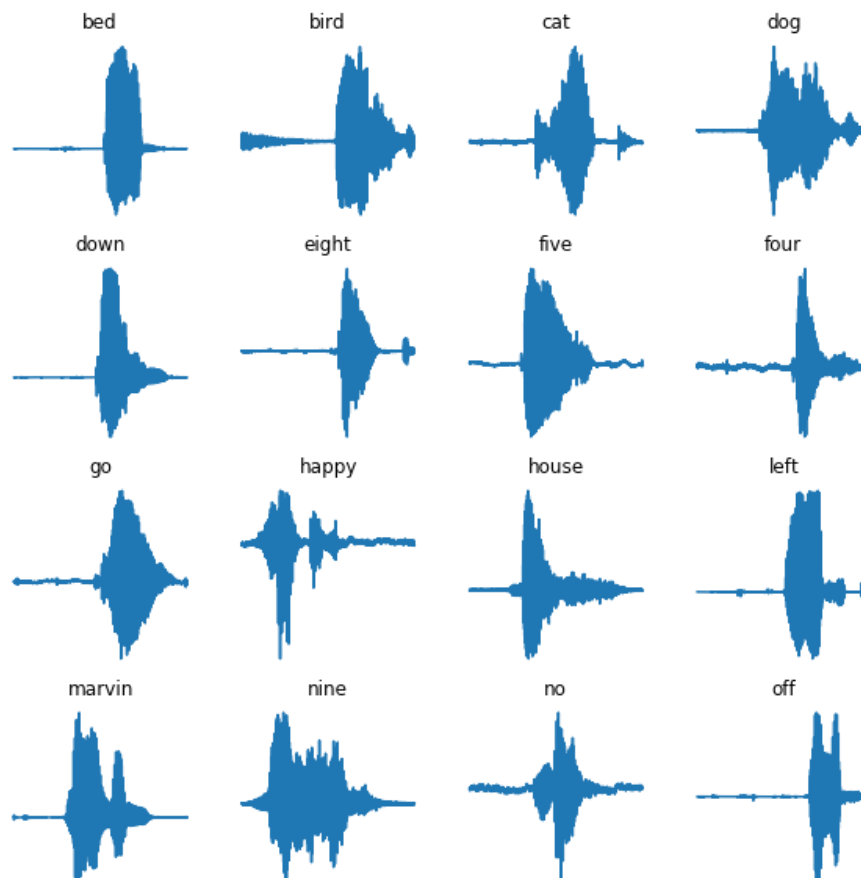
输出结果：



步骤 11 多样本单形可视化

代码如下：

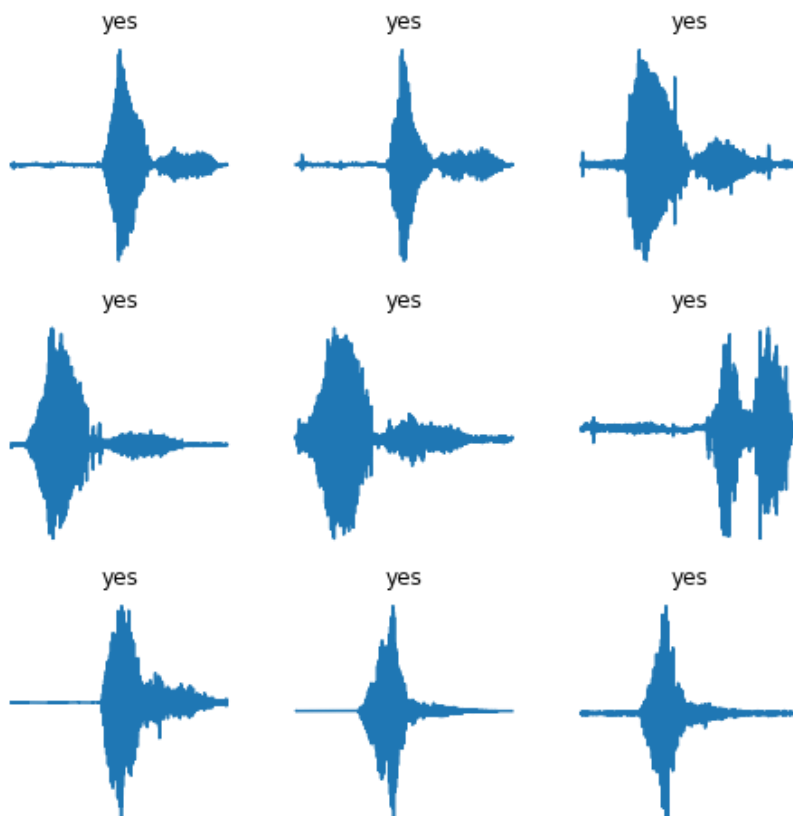
```
fig = plt.figure(figsize=(10,10))
for i, filepath in enumerate(sample_audio[:16]):
    plt.subplot(4,4,i+1)
    samplerate, test_sound = wavfile.read(filepath)
    plt.title(filepath.split('/')[-2])
    plt.axis('off')
    plt.plot(test_sound)
plt.show()
输出结果：
```



步骤 12 单样本多波形可视化

代码如下：

```
fig = plt.figure(figsize=(8,8))
for i, filepath in enumerate(yes_samples):
    plt.subplot(3,3,i+1)
    samplerate, test_sound = wavfile.read(filepath)
    plt.title(filepath.split('/')[-2])
    plt.axis('off')
    plt.plot(test_sound)
plt.show()
输出结果：
```



1.3 思考题

- 如何对上述数据做分帧、静音切除等预处理操作？
- 如何对上述数据做数增强？

1.4 实验小结

本实验是基于 Python 语言和语音处理框架 wave，以及开源数据集进行的语音数据预处理操作实验，主要包括了对语音数据基础信息查看，波形文件，频谱文件的处理等，通过可视化和具体数值的显示更加直观的查看语音数据的本质属性。

2 语音合成和语音识别

2.1 实验介绍

2.1.1 关于本实验

语音交互服务（Speech Interaction Service，简称 SIS）是一种人机交互方式，用户通过实时访问和调用 API（Application Programming Interface，应用程序编程接口）将语音识别成文字或者将文本转换成逼真的语音等。

本实验将基于华为云语音交互服务实现语音识别及语音合成功能。

2.1.2 实验目的

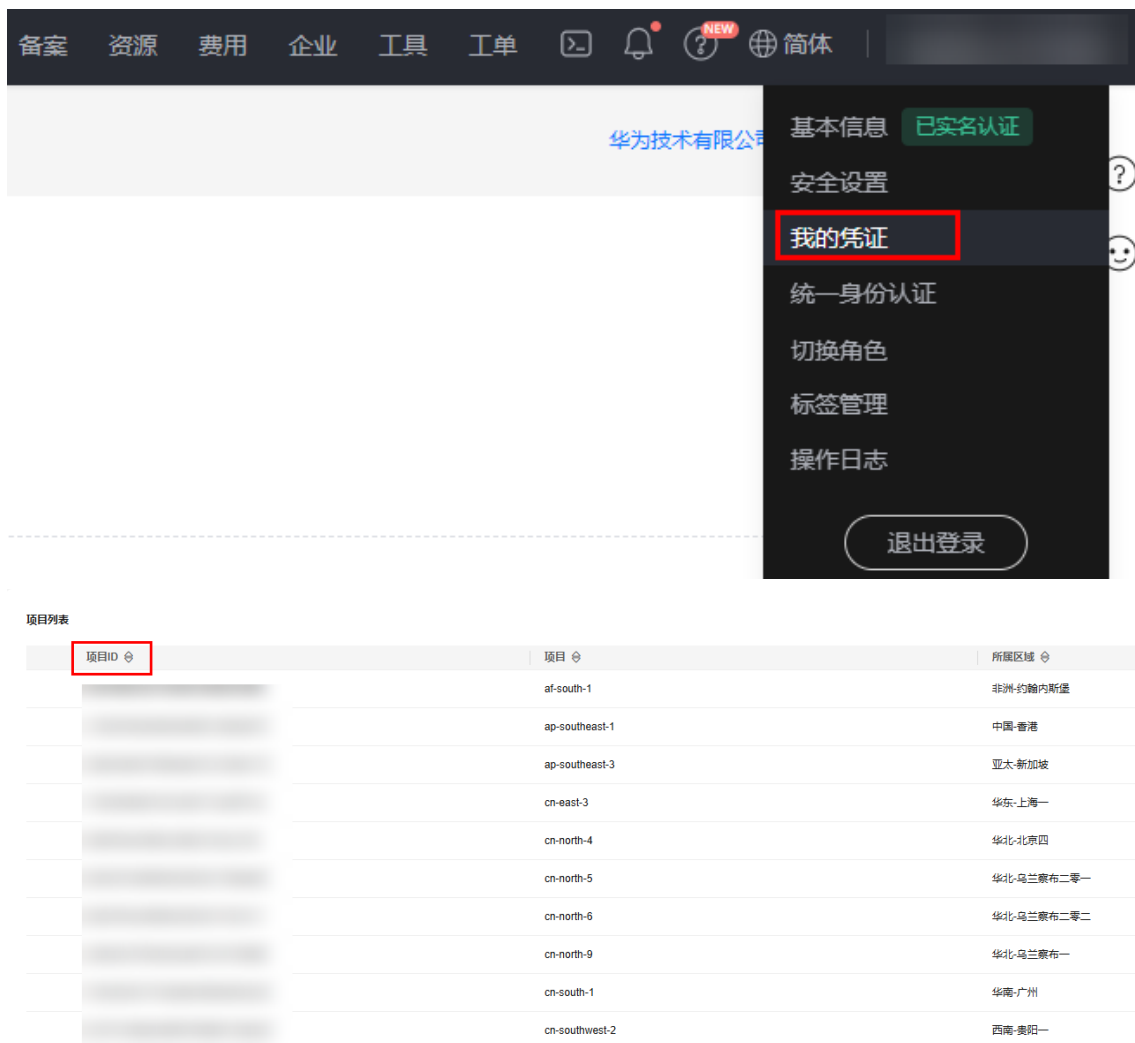
- 了解如何结合华为云进行语音合成与语音识别的功能。
- 理解和掌握如何使用 Python 进行该业务的开发。

2.1.3 实验环境准备

注册并登录华为云管理控制台。

了解语音合成与语音识别相关文档，详见 https://support.huaweicloud.com/sdkreference-sis/sis_05_0051.html 和 https://support.huaweicloud.com/sdkreference-sis/sis_05_0055.html。

准备华为云账号的 project_id。登录华为云，在用户名处点击“我的凭证”。



The screenshot shows the Huawei Cloud IAM console. The top navigation bar includes links for 备案, 资源, 费用, 企业, 工具, 工单, and a search bar. The user's profile is visible as 华为技术有限公司. The '我的凭证' (My Credentials) menu is highlighted, showing options like 基本信息 (Basic Information), 安全设置 (Security Settings), 我的凭证 (My Credentials), 统一身份认证 (Unified Identity Authentication), 切换角色 (Switch Role), 标签管理 (Tag Management), and 操作日志 (Operation Log). Below the menu, the '项目列表' (Project List) is displayed, showing a table of projects with columns for 项目ID (Project ID), 项目 (Project), and 所属区域 (Region).

项目ID	项目	所属区域
af-south-1	非洲-约翰内斯堡	非洲-约翰内斯堡
ap-southeast-1	中国-香港	中国-香港
ap-southeast-3	亚太-新加坡	亚太-新加坡
cn-east-3	华东-上海一	华东-上海一
cn-north-4	华北-北京四	华北-北京四
cn-north-5	华北-乌兰察布二零一	华北-乌兰察布二零一
cn-north-6	华北-乌兰察布二零二	华北-乌兰察布二零二
cn-north-9	华北-乌兰察布一	华北-乌兰察布一
cn-south-1	华南-广州	华南-广州
cn-southwest-2	西南-贵阳一	西南-贵阳一

准备 AK/SK（如果已获取，可以直接使用）。在“我的凭证”界面，选择“管理访问密钥 > 新增访问密钥”来获取，下载认证账号的 AK/SK，请妥善保管 AK/SK 信息。之后可以直接使用此 AK/SK 信息。



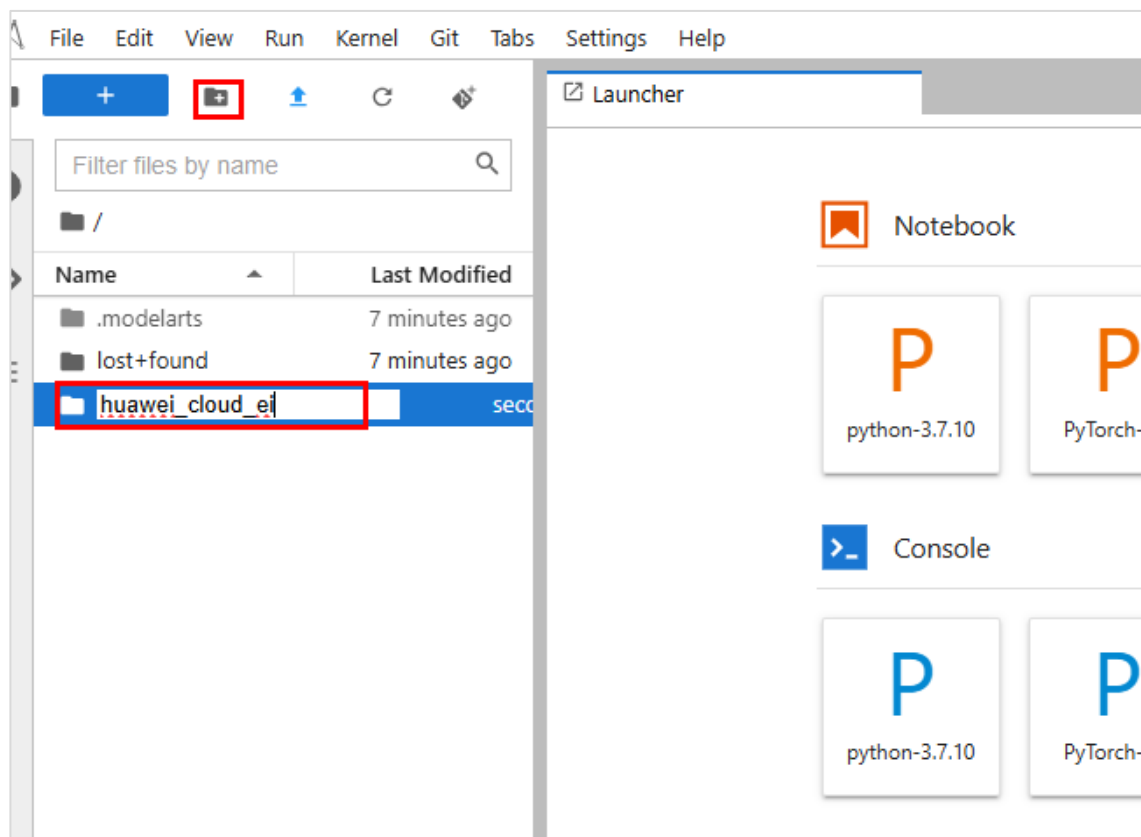
The screenshot shows the '我的凭证' (My Credentials) page. The 'API凭证' (API Credentials) section is selected, and the '访问密钥' (Access Key) sub-section is highlighted. The '访问密钥' section contains a warning message: '如果访问密钥泄露，会带来数据泄露风险，且每个访问密钥仅能下载一次。为了账号安全性，建议您定期更换并妥善保存访问密钥。' (If the access key is leaked, it will bring data leakage risk, and each access key can only be downloaded once. For account security, we recommend you to regularly replace and properly save the access key.) Below the warning, there is a button to '新增访问密钥' (Add Access Key). The table below shows the list of access keys.

访问密钥ID	描述	状态
...	...	启用

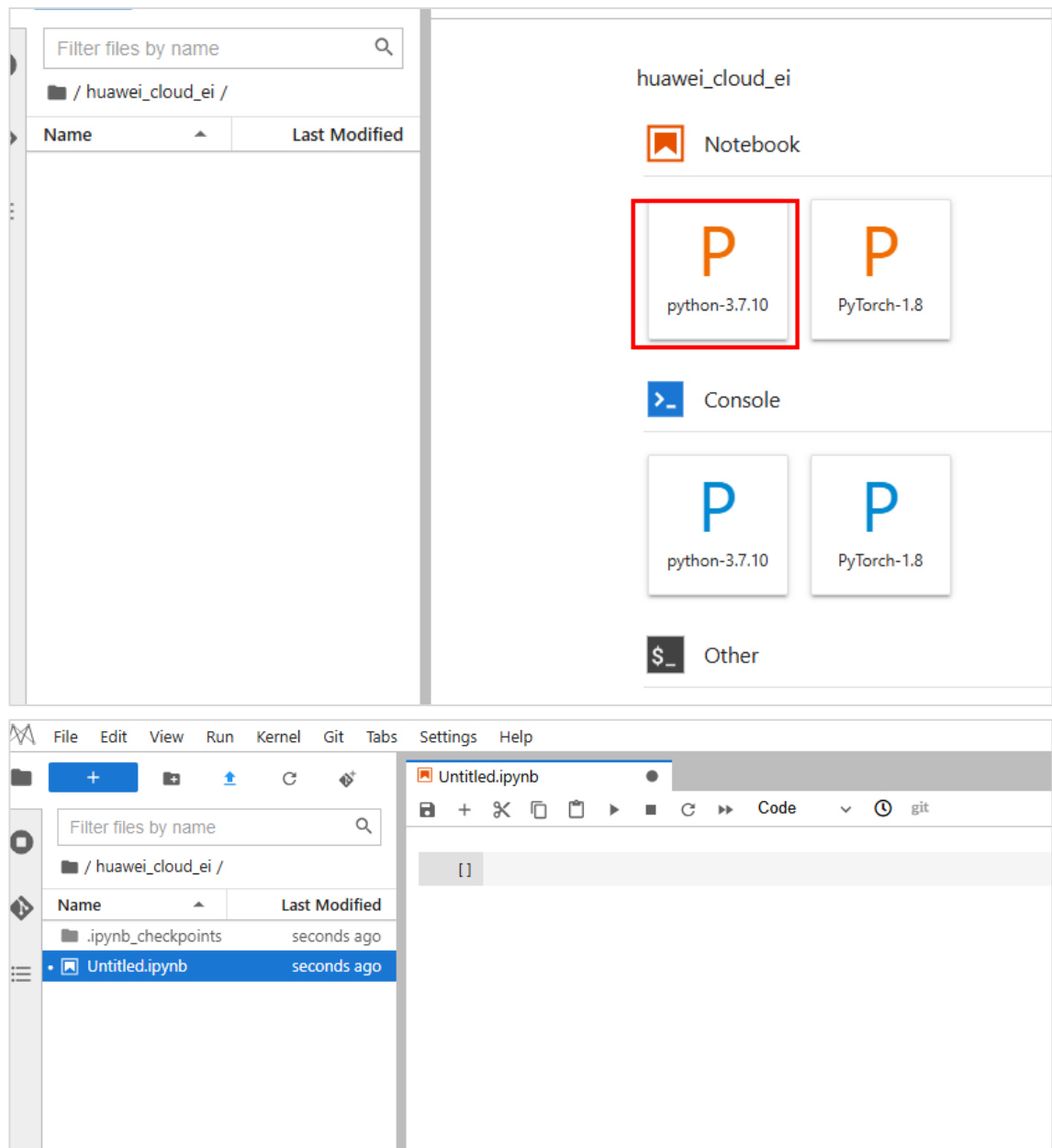
2.1.4 SDK 安装

步骤 1 进入创建好的 ModelArts Python notebook 环境

步骤 2 进入 notebook 界面，新建文件夹，并重命名文件夹为 huawei_cloud_ei



步骤 3 点击进入刚创建的 huawei_cloud_ei 文件夹，新建 notebook 文件，选择 python3 环境



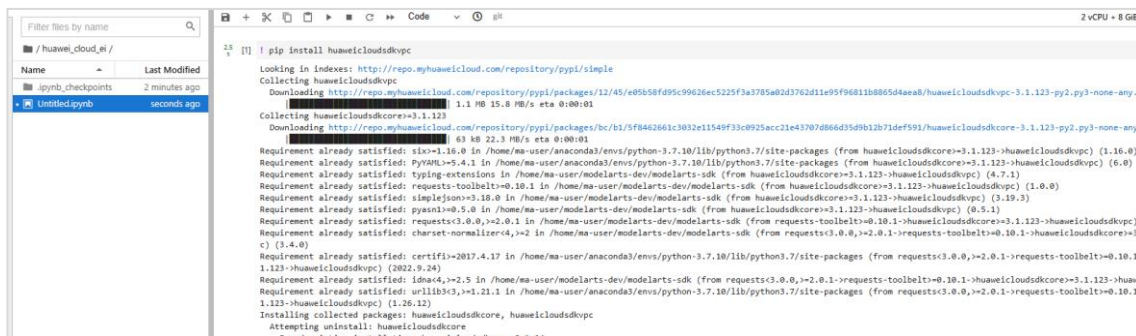
步骤 4 安装 Python 版本 SDK

安装 VPC 服务包。

输入：

```
! pip install huaweicloudsdkvpc
```

输出：



步骤 5 通过命令安装语音交互服务 SDK (Python3)

```
!pip install huaweicloudsdksis
!pip install huaweicloudsdkcore
```

2.2 实验步骤

2.2.1 语音合成

定制语音合成，是一种将文本转换成逼真语音的服务。用户通过实时访问和调用 API 获取语音合成结果，将用户输入的文字合成为音频。

步骤 1 导入所需要的包

```
# -*- coding: utf-8 -*-
import os
from huaweicloudsdkcore.auth.credentials import BasicCredentials
from huaweicloudsdksis.v1.region.sis_region import SisRegion
from huaweicloudsdkcore.exceptions import exceptions
from huaweicloudsdksis.v1 import *
```

步骤 2 配置相关参数

```
ak = "" #配置自己的 ak
sk = "" #配置自己的 sk
project_id = "" #配置自己的 project_id
```

步骤 3 初始化客户端

```
def init_client(ak: str = "", sk: str = "", project_id: str = "") -> object:
    credentials = BasicCredentials(ak = ak, sk = sk, project_id = project_id) # 身份认证
    # 初始化客户端
    client = SisClient.new_builder() \
        .with_credentials(credentials) \
        .with_region(SisRegion.value_of("cn-north-4")) \
```

```
.build() ## 当前为北京四，可以根据所在区域更换 cn-north-4
return client
client = init_client(ak=ak, sk=sk, project_id=project_id)
```

步骤 4 构造并发送请求

```
def tts(text: str = ""):

    try:
        request = RunTtsRequest()
        configbody = TtsConfig(
            audio_format="wav", # 生成音频格式
            sample_rate="16000", # 音频采样率
            _property="chinese_xiaoqi_common", # 音色
            speed=0, # 语速
            pitch=0, # 音高
            volume=50, # 音量
        )

        request.body = PostCustomTTSReq(
            config=configbody,
            text=text
        )

        response = client.run_tts(request)
        return response
    except exceptions.ClientRequestException as e:
        print(e.status_code)
        print(e.request_id)
        print(e.error_code)
        print(e.error_msg)
        return None

res = tts(text="你这猴子真是令我欢喜")
```

返回对象 res 中包含以下信息：

```
{
  "result": {
    "data": "UklGRuT...
    ...
  },
  "trace_id": "b9295ebb-1c9c-4d00-b2e9-7d9f3dd63727",
  .....
}
```

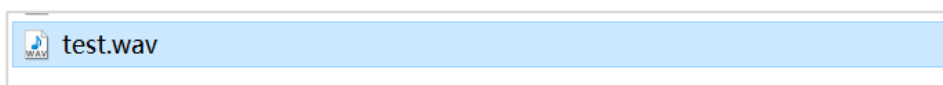
trace_id 代表服务内部的令牌，可用于在日志中追溯具体流程，调用失败无此字段，在某些错误情况下可能没有此令牌字符串；result 代表调用成功表示识别结果，调用失败时无此字段。音频信息以 base64 编码格式返回。

步骤 5 保存合成音频

```
import base64
import json
with open("test.wav","wb") as f:

    f.write(base64.b64decode(res.result.data))
```

保存的语音数据如下：



2.2.2 语音识别

一句话识别接口，用于短语音的同步识别。一次性上传整个音频，响应中即返回识别结果。

步骤 1 导入所需要的包

```
# -*- coding: utf-8 -*-
import os
from huaweicloudsdkcore.auth.credentials import BasicCredentials
from huaweicloudsksis.v1.region.sis_region import SisRegion
from huaweicloudsdkcore.exceptions import exceptions
from huaweicloudsksis.v1 import *
```

步骤 2 配置相关参数

```
ak = "****" #配置自己的 ak
sk = "****" #配置自己的 sk
project_id = "****" #配置自己的 project_id
```

步骤 3 配置数据和属性

```
# 一句话识别参数，我们使用语音合成的语音数据，1min 以内的音频
path = 'test.wav'
path_audio_format = 'wav' # 音频格式，详见 api 文档
path_property = 'chinese_16k_common' # language_sampleRate_domain, 如 chinese_8k_common, 详见 api 文档
```

步骤 4 初始化客户端

```
def init_client(ak: str = "", sk: str = "", project_id: str = "")-> object:
    credentials = BasicCredentials(ak = ak , sk = sk ,project_id = project_id) # 身份认证
    # 初始化客户端
```

```
client = SisClient.new_builder() \
    .with_credentials(credentials) \
    .with_region(SisRegion.value_of("cn-north-4")) \
    .build() # # 当前为北京四，可以根据所在区域更换 cn-north-4
return client
```

步骤 5 构造请求

```
import base64

def asr(path: str="", path_audio_format: str="wav", path_property: str="chinese_16k_general"):
    """
    path: 音频文件路径
    path_audio_format: 格式，如 wav, mp3,m4a 等，也可设置为 auto
    path_property: 所使用的模型特征串。通常是“语种_采样率_领域”的形式

    """
    with open(path, 'rb') as f:
        audio = base64.b64encode(f.read())# 获取音频文件的 b64 编码

    try:
        request = RecognizeShortAudioRequest()
        configbody = Config(
            audio_format=path_audio_format,
            _property=path_property
        )
        request.body = PostShortAudioReq(
            data=audio, # 此处接受的是音频文件的 base64 编码
            config=configbody
        )
        response = client.recognize_short_audio(request)
        return response
    except exceptions.ClientRequestException as e:
        print(e.status_code)
        print(e.request_id)
        print(e.error_code)
        print(e.error_msg)
        return None

res = asr(path=path, path_audio_format=path_audio_format, path_property=path_property)
print(res)
```

步骤 6 输出结果：

```
{"trace_id": "36d55614-da1f-497c-b61a-3969bf6d2bb6", "result": {"text": "你这猴子真是令我欢喜",
"score": 0.9149643778800964}}
```

trace_id 代表服务内部的令牌，可用于在日志中追溯具体流程，调用失败无此字段，在某些错误情况下可能没有此令牌字符串；result 代表调用成功表示识别结果，调用失败时无此字段；text 代表调用成功表示识别出的内容；score 代表调用成功表示识别出的置信度（0-1 之间）。

2.3 实验小结

本章主要介绍了应用华为公有云上的语音交互服务中（语音合成和语音识别）进行实验的具体操作，主要是通过 SDK 发布 RestFul 请求进行相关功能的实现，而在使用 SDK 发布 RestFul 请求时，需要借助进行必要用户认证信息的配置，在本章中主要针对 AK\SK 的方式进行了系统的介绍和说明，帮助学员使用语音合成和语音识别提供了实际的操作指导。

3 基于 Seq2Seq 的语音识别

3.1 实验介绍

3.1.1 关于本实验

本实验旨在利用基于循环神经网络（RNN）的 seq2seq 模型实现语音识别功能。

实验将通过 pytorch 框架搭建模型结构，包括编码器、解码器及注意力机制等核心组件。使用预处理后的语音数据作为输入，通过模型将语音信号转换为文本序列。

3.1.2 实验目的

- 掌握通过 pytorch 2.5 搭建 Seq2Seq 模型。
- 掌握应用 Seq2Seq 模型实现语音识别。

3.1.3 实验预备知识

本实验操作需要具备 3 方面的预备知识：

- 有 Seq2Seq 的理论基础。
- 有 pytorch 的编程基础。

3.2 实验步骤

步骤 1 导入相关模块

下载数据以及配置环境，代码如下（数据为 speech\实验 3-深度学习代码和数据，若已下载可跳过此步骤）：

```
#在北京四以外的其他区域的 notebook 执行上述代码出现数据下载超时，添加下方网络环境配置。
%env no_proxy=repo.myhuaweicloud.com,modelarts.cn-north-9.myhuaweicloud.com,obs.cn-north-9.myhuaweicloud.com,iam.cn-north-9.huaweicloud.com,iam-cache-proxy.cn-north-9.myhuaweicloud.com,dli.cn-north-9.myhuaweicloud.com,127.0.0.1,localhost,172.16.*
%env NO_PROXY=repo.myhuaweicloud.com,modelarts.cn-north-9.myhuaweicloud.com,obs.cn-north-9.myhuaweicloud.com,iam.cn-north-9.huaweicloud.com,iam-cache-proxy.cn-north-9.myhuaweicloud.com,dli.cn-north-9.myhuaweicloud.com,127.0.0.1,localhost,172.16.*

#下载数据
```

```
! wget https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIP-AI%20EI%20Developer/V2.5/chapter3/speech.zip
```

解压下载好的文件压缩包，代码如下：

```
! unzip speech.zip
```

环境配置：

```
! pip install python_speech_features
```

导入相关包代码如下：

```
# coding=utf-8
import warnings
warnings.filterwarnings("ignore")
import time
import torchaudio
import torch
import torch.nn as nn
import numpy as np
from python_speech_features import mfcc
```

步骤 2 配置数据路径

代码如下（数据通过实验 1 解压的“实验 3-深度学习代码和数据”的 data 文件夹）：

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

步骤 3 读取数据并进行特征抽取

代码如下：

```
# 定义稀疏张量的函数
def sparse_tuple_from(sequences, dtype=torch.long):
    indices = []
    values = []

    # 创建一个映射字典，将每个唯一的字符串映射到一个唯一的整数
    label_map = {label: idx for idx, label in enumerate(set(sequences[0]))}

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

    indices = torch.tensor(indices, dtype=dtype)
    values = torch.tensor(values, dtype=dtype)
    shape = torch.tensor([len(sequences), indices.max(0)[0][1] + 1], dtype=dtype)

    return indices, values, shape
```

```
# 读取音频文件并提取 MFCC 特征
def get_audio_feature(audio_filename):
    # 读取 wav 文件内容, fs 为采样率, audio 为数据
    waveform, sample_rate = torchaudio.load(audio_filename)

    # 确保音频是单通道
    if waveform.shape[0] > 1:
        waveform = waveform.mean(dim=0, keepdim=True)

    # 提取 mfcc 特征
    inputs = mfcc(waveform.numpy()[0], samplerate=sample_rate)
    # 对特征数据进行归一化, 减去均值除以方差
    feature_inputs = torch.tensor(inputs, dtype=torch.float32).unsqueeze(0)
    feature_inputs = (feature_inputs - feature_inputs.mean()) / feature_inputs.std()

    # 特征数据的序列长度
    feature_seq_len = [feature_inputs.shape[1]]
    return feature_inputs, feature_seq_len

# 读取标签文件并转换为稀疏张量
def get_audio_label(target_filename):
    with open(target_filename, 'r') as f:
        # 原始文本为“i like you , do you like me”
        line = f.readlines()[0].strip()
    # 放入 list 中, 空格用'代替
    # ['i', ' ', 'like', ' ', 'you', ' ', ' ', ' ', 'do', ' ', 'you', ' ', 'like', ' ', 'me']
    targets = line.split(' ')
    targets.insert(0, '<START>')
    targets.append("<END>")
    print(targets)
    # 将列表转换成稀疏三元组
    train_targets = sparse_tuple_from([targets])
    return targets, train_targets

feature_inputs, feature_seq_len = get_audio_feature(audio_filename)
line_targets, train_target = get_audio_label(target_filename)

print("Feature Inputs Shape:", feature_inputs.shape)
print("Feature Sequence Length:", feature_seq_len)
print("Line Targets:", line_targets)
print("Train Target Indices:", train_target[0])
print("Train Target Values:", train_target[1])
print("Train Target Shape:", train_target[2])
```

输出如下:

```
[<START>, 'i', 'like', 'you', ' ', 'do', 'you', 'like', 'me', '<END>']
Feature Inputs Shape: torch.Size([1, 333, 13])
```

```
Feature Sequence Length: [333]
Line Targets: ['<START>', 'i', 'like', 'you', ',', 'do', 'you', 'like', 'me', '<END>']
Train Target Indices: tensor([[0, 0],
                               [0, 1],
                               [0, 2],
                               [0, 3],
                               [0, 4],
                               [0, 5],
                               [0, 6],
                               [0, 7],
                               [0, 8],
                               [0, 9]])
Train Target Values: tensor([6, 7, 0, 3, 5, 4, 3, 0, 2, 1])
Train Target Shape: tensor([ 1, 10])
```

步骤 4 神经网络参数配置

代码如下：

```
target_characters = list(set(line_targets))
INPUT_LENGTH = feature_inputs.shape[-2]
OUTPUT_LENGTH = train_target[-1][-1]
# 输入特征维度
INPUT_FEATURE_LENGTH = feature_inputs.shape[-1]
# 输出特征维度（目标字符数量）
OUTPUT_FEATURE_LENGTH = len(target_characters)
# LSTM 单元数量
N_UNITS = 256
# 批量大小
BATCH_SIZE = 1
EPOCH = 40
NUM_SAMPLES = 1
target_texts = []
target_texts.append(line_targets)
```

步骤 5 创建 Seq2Seq 模型

代码如下：

```
import torch.optim as optim

class Encoder(nn.Module):
    def __init__(self, n_input, n_units):
        super(Encoder, self).__init__()
        self.lstm = nn.LSTM(input_size=n_input, hidden_size=n_units, batch_first=True)

    def forward(self, x):
        # x: [batch_size, seq_length, n_input]
        _, (hidden, cell) = self.lstm(x) # 只取最后的状态
```

```
        return hidden, cell

class Decoder(nn.Module):
    def __init__(self, n_output, n_units):
        super(Decoder, self).__init__()
        self.lstm = nn.LSTM(input_size=n_output, hidden_size=n_units, batch_first=True)
        self.fc = nn.Linear(n_units, n_output)

    def forward(self, x, hidden, cell):
        # x: [batch_size, seq_length, n_output]
        output, (hidden, cell) = self.lstm(x, (hidden, cell))
        output = self.fc(output)
        return output, hidden, cell

class Seq2Seq(nn.Module):
    def __init__(self, n_input, n_output, n_units):
        super(Seq2Seq, self).__init__()
        self.encoder = Encoder(n_input, n_units)
        self.decoder = Decoder(n_output, n_units)

    def forward(self, encoder_input, decoder_input):
        # 编码器
        hidden, cell = self.encoder(encoder_input)

        # 解码器
        decoder_output, _, _ = self.decoder(decoder_input, hidden, cell)
        return decoder_output

    def inference(self, encoder_input, max_length, sos_token):
        # 推理阶段
        hidden, cell = self.encoder(encoder_input)
        batch_size = encoder_input.size(0)
        decoder_input = torch.tensor([[sos_token]], dtype=torch.long).repeat(batch_size, 1)

        outputs = []
        for _ in range(max_length):
            output, hidden, cell = self.decoder(decoder_input, hidden, cell)
            output = torch.argmax(output, dim=-1)
            outputs.append(output)
            decoder_input = output

        return torch.cat(outputs, dim=1)

# 创建模型
```

```
model = Seq2Seq(n_input=INPUT_FEATURE_LENGTH, n_output=OUTPUT_FEATURE_LENGTH,
n_units=N_UNITS)

# 定义优化器和损失函数
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

# 打印模型结构
print(model)
```

输出结果：

```
Seq2Seq(
  (encoder): Encoder(
    (lstm): LSTM(13, 256, batch_first=True)
  )
  (decoder): Decoder(
    (lstm): LSTM(8, 256, batch_first=True)
    (fc): Linear(in_features=256, out_features=8, bias=True)
  )
)
```

步骤 6 配置训练数据

代码如下：

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

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

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

输出结果：

```
(1, 10, 8)
0 <START>
1 i
2 like
3 you
4 ,
5 do
```

```
6 you
7 like
8 me
9 <END>
```

步骤 7 模型训练

代码如下：

```
char_to_index = {char: idx for idx, char in enumerate(target_characters)}
target_indices = torch.tensor([[char_to_index[char] for char in line_targets]], dtype=torch.long)

model.train()
for epoch in range(EPOCH):
    optimizer.zero_grad()

    # 输入数据和目标数据
    encoder_input = feature_inputs # 输入特征 [batch_size, seq_length, input_feature_length]
    decoder_input = torch.nn.functional.one_hot(target_indices[:, :-1],
num_classes=OUTPUT_FEATURE_LENGTH).float() # 解码器输入 [batch_size, seq_length,
output_feature_length]
    target = target_indices[:, 1:] # 真实目标 [batch_size, seq_length]

    # 前向传播
    output = model(encoder_input, decoder_input)

    # 计算损失
    loss = criterion(output.transpose(1, 2), target)
    loss.backward()
    optimizer.step()

    print(f"Epoch {epoch + 1}/{EPOCH}, Loss: {loss.item()}")
```

输出结果：

Train on 1 samples

```
Epoch 1/40, Loss: 2.078908920288086
Epoch 2/40, Loss: 2.0622339248657227
Epoch 3/40, Loss: 2.045137405395508
Epoch 4/40, Loss: 2.0260937213897705
.....
Epoch 37/40, Loss: 0.29518258571624756
Epoch 38/40, Loss: 0.26819998025894165
Epoch 39/40, Loss: 0.24317796528339386
Epoch 40/40, Loss: 0.2207832783460617
```

步骤 8 模型测试

代码如下：

```
def predict_chinese(source, encoder_inference, decoder_inference, n_steps, target_dict,
target_dict_reverse):
    # 先通过推理 encoder 获得预测输入序列的隐状态
    with torch.no_grad(): # 推理阶段不需要计算梯度
        state = encoder_inference(source) # encoder_inference 是 Encoder 模型的推理方法

    # 第一个字符 '<START>', 为起始标志
    predict_seq = torch.tensor([[target_dict['<START>']], dtype=torch.long) # 初始化为起始字符索引

    output = "" # 用于存储最终的预测结果
    for i in range(n_steps): # n_steps 为句子最大长度
        # 将索引转换为 one-hot 编码
        predict_seq_one_hot = torch.nn.functional.one_hot(predict_seq,
num_classes=OUTPUT_FEATURE_LENGTH).float()

        # 给 decoder 输入上一个时刻的 h, c 隐状态, 以及上一次的预测字符 predict_seq
        yhat, h, c = decoder_inference(predict_seq_one_hot, *state) # 推理 decoder

        # 获取预测结果
        char_index = yhat.argmax(dim=-1).item() # 获取预测字符的索引
        char = target_dict_reverse[char_index] # 将索引转换为字符

        # 更新状态
        state = (h, c) # 本次状态作为下一次的初始状态继续传递
        predict_seq = torch.tensor([[char_index]], dtype=torch.long) # 更新预测序列

        if char == '<END>': # 预测到了终止符则停下来
            break
        output += " " + char # 将预测的字符添加到输出中

    return output.strip() # 返回预测结果并去除首尾空格

# 示例调用
out = predict_chinese(
    source=feature_inputs, # 输入特征
    encoder_inference=model.encoder, # Encoder 模型
    decoder_inference=model.decoder, # Decoder 模型
    n_steps=OUTPUT_LENGTH, # 最大预测长度
    target_dict=target_dict, # 字符到索引的映射
    target_dict_reverse=target_dict_reverse # 索引到字符的映射
)

print(out)
```

输出结果:

i like you , do you like me

备注：本实验只用了一个训练样本，有兴趣的同学可以进一步将模型扩展到更多样本空间上训练。另外，每次训练得到的模型，因参数不同，在预测时输出的结果可能会有差异。

3.3 思考题

- 如何将模型应用到自身的数据集上？
- 如何进一步提升模型的准确率？

3.4 实验小结

在本次实验中，构建了基 RNN 的 seq2seq 语音识别模型，并利用 pytorch 框架实现了模型训练与测试。

华为认证 AI 系列教程

HCIP-AI-EI Developer

华为企业AI开发高级工程师认证

自然语言处理

实验指导手册

版本:2.5



华为技术有限公司

版权所有 © 华为技术有限公司 2026。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<https://e.huawei.com>

华为认证体系介绍

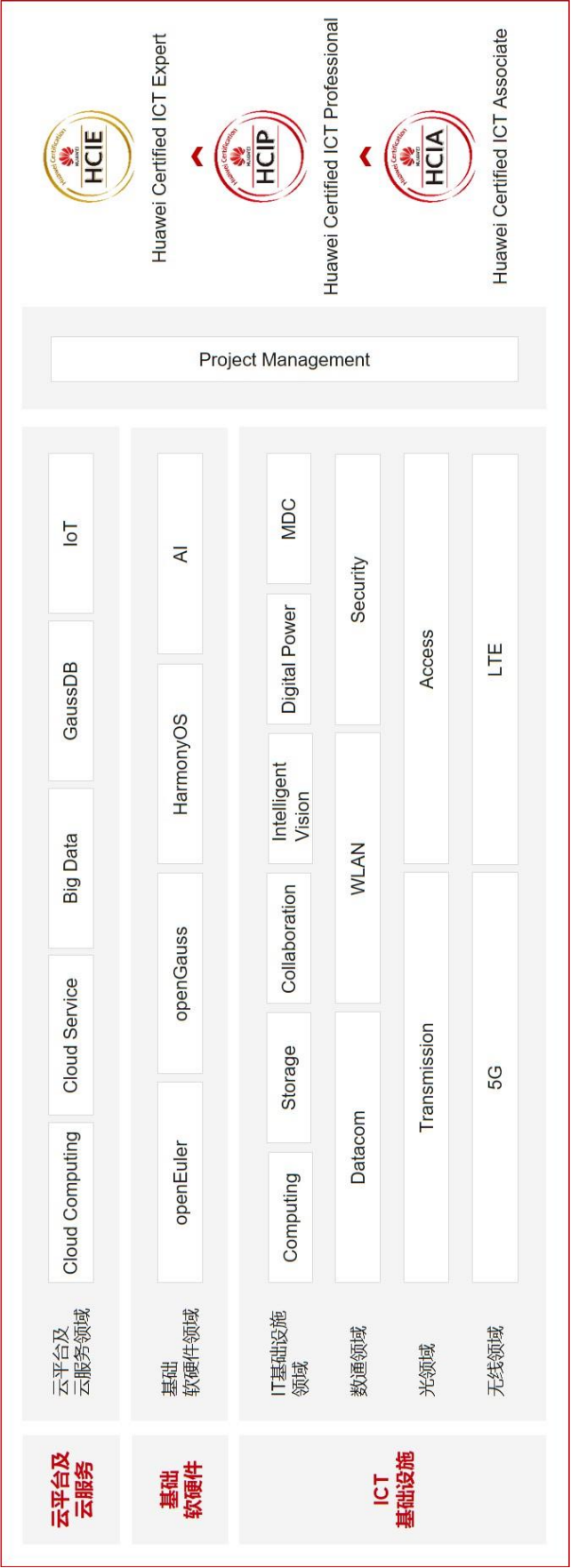
华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT基础设施认证、基础软硬件认证、云平台及云服务认证三类认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIP-AI-EI Developer V2.5定位于培养与认证具有图像处理、语音处理、自然语言处理能力，能够应用通用开源框架和华为云ModelArts平台等进行开发和创新的高级工程师。

通过HCIP-AI-EI Developer V2.5认证，您将掌握神经网络基础、图像处理理论、语音处理理论、自然语言处理理论、大模型深度学习算法等基础知识，具备运用通用开源框架和华为云ModelArts平台等进行图像处理、语音处理和自然语言处理等人工智能产品设计开发的能力，能够胜任AI开发高级工程师、图像处理开发工程师、语音处理开发工程师、自然语言处理开发工程师等岗位。



简介

本书为 HCIP-AI 认证培训教程，适用于准备参加 HCIP-AI 考试的学员或者希望了解 AI 基础知识的读者。掌握本实验手册内容，您将能够使用 ModelArts 一站式开发平台搭建并训练自己的自然语言处理相关的算法模型。

内容描述

本实验指导书共包含 3 组实验，涉及到自然语言处理基础算法，自然语言理解，自然语言生成等内容。

- 实验 1 文本分类
- 实验 2 机器翻译

读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基本的 Python 语言编辑能力
- 有一定自然语言处理的理论基础
- 对 PyTorch 框架一定的了解

实验环境说明

实验使用语言为 Python，框架为 PyTorch。

- Python3.9 及以上
 - torch>=2.1 (GPU、cuda11.8/12.4)
-



目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
实验环境说明	3
1 文本分类	5
1.1 实验简介	5
1.2 实验目的	5
1.3 实验步骤	5
1.3.1 环境准备	5
1.3.2 朴素贝叶斯文本分类	6
1.3.3 SVM 文本分类	9
1.3.4 TextCNN 文本分类	13
1.3.5 思考题	20
1.4 思考题	20
1.5 实验小结	20
2 机器翻译	21
2.1 实验简介	21
2.2 实验环境	21
2.3 实验步骤	21
2.4 实验小结	35
2.5 思考题	35

1 文本分类

1.1 实验简介

本节实验介绍如何实现一个文本分类模型，具体任务为用户评论的情感分析，方法包括：

- 朴素贝叶斯文本分类
- SVM 文本分类
- TextCNN 文本分类

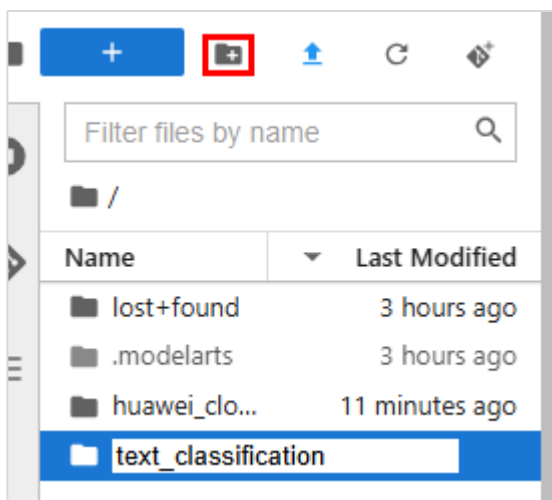
1.2 实验目的

- 理解文本分类任务的基本原理及流程
- 理解朴素贝叶斯、SVM、CNN 算法之间的差异
- 掌握基于 PyTorch2.5 版本构建深度学习网络的方法

1.3 实验步骤

1.3.1 环境准备

步骤 1 进入 notebook 界面，新建文件夹，并重命名文件夹为 text_classification



步骤 2 点击进入刚创建的 text_classification 文件夹

步骤 3 新建 notebook 文件，选择 PyTorch 环境

步骤 4 下载数据

输入：

```
! https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIP-AI%20EI%20Developer/V2.5/chapter4/nlp.zip
```

步骤 5 解压数据

输入：

```
! unzip nlp.zip
```

步骤 6 安装依赖包

输入：

```
! pip install jieba
```

输出：

```
! pip install jieba
Looking in indexes: http://repo.myhuaweicloud.com/repository/pypi/simple
Collecting jieba
  Downloading http://repo.myhuaweicloud.com/repository/pypi/packages/c6/cb/18eeb235f833b726522d7ebd54f2278ce28ba9438e3135ab0278d9792a2/jieba-0.42.1.tar.gz (19.2 MB)
    [REDACTED] 19.2 MB 19.5 MB/s eta 0:00:01
Building wheels for collected packages: jieba
  Building wheel for jieba (setup.py) ... done
  Created wheel for jieba: filename=jieba-0.42.1-py3-none-any.whl size=19314458 sha256=ad794247144e41d48aee9b692bae4cede2d7ceb47c261593594edb5838415cc1
  Stored in directory: /home/ma-user/.cache/pip/wheels/86/bc/97/67c05f24b07573fd425039f944f8bc57c8db6f2c0bcc046fa
Successfully built jieba
Installing collected packages: jieba
Successfully installed jieba-0.42.1
WARNING: You are using pip version 21.0.1; however, version 24.0 is available.
You should consider upgrading via the '/home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python3.7 -m pip install --upgrade pip' command.
```

1.3.2 朴素贝叶斯文本分类

步骤 1 导入相关库

输入：

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

步骤 2 加载数据

输入：

```
train_data = pd.read_csv('data/chnsenticorp/train.tsv', sep='\\t')
```

```
test_data = pd.read_csv('data/chnsenticorp/test.tsv', sep='\t')
label_map = {0: '负面', 1: '正面'}
x_train, y_train = train_data.text_a.values, train_data.label.values
x_test, y_test = test_data.text_a.values, test_data.label.values
```

数据形式如下所示：

	label	text_a
1	1	选择珠江花园的原因就...
2	1	15.4寸笔记本的键盘确...
3	0	房间太小。其他的都一...
4	0	1.接电源没有几分钟,...
5	1	今天才知道这书还有第...
6	0	机器背面似乎被撕了张...
7	0	呵呵，虽然表皮看上去...
8	0	这本书实在是太烂了,...
9	1	地理位置佳，在市中心...
10	1	5.1期间在这住的，位...
11	0	我看过朋友的还可以，...
12	1	还不错，设施稍微有点...
13	1	轻便，方便携带，性能...
14	1	入住的是度假区的豪华...
15	0	1.模具具有摄像头的位置...
16	0	送的内胆包有点不好，...
17	1	这是我第1次给全五星...
18	0	很好的地理位置，一踢...
19	1	漂亮（老婆认为的），...
20	0	内存太小，偶配了2根...

步骤 3 定义分类器主类，定义训练和测试函数

输入：

```
class NB_Classifier(object):

    def __init__(self):
        # 朴素贝叶斯分类器
        self.model = MultinomialNB(alpha=1) #拉普拉斯平滑：1
        # 使用 tf-idf 特征提取
        self.feature_processor = TfidfVectorizer(tokenizer=jieba.cut)

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

```
# tf-idf 特征提取
x_train_fea = self.feature_processor.fit_transform(x_train)
self.model.fit(x_train_fea, y_train)

train_accuracy = self.model.score(x_train_fea, y_train)
print("训练集准确率: {}".format(round(train_accuracy, 3)))

x_test_fea = self.feature_processor.transform(x_test)
y_predict = self.model.predict(x_test_fea)
test_accuracy = accuracy_score(y_test, y_predict)
print("测试集准确率: {}".format(round(test_accuracy, 3)))

y_predict = self.model.predict(x_test_fea)
print('测试集评估: ')
print(classification_report(y_test, y_predict, target_names=['0', '1']))

def single_predict(self, text):
    text_preprocess = [" ".join(jieba.cut(text))]
    text_fea = self.feature_processor.transform(text_preprocess)
    predict_idx = self.model.predict(text_fea)[0]
    predict_label = label_map[predict_idx]
    predict_prob = self.model.predict_proba(text_fea)[0][predict_idx]

    return predict_label, predict_prob
```

步骤 4 初始化分类器，并训练

输入：

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

测试集评估结果如下：

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

```
Building prefix dict from the default dictionary ...
Dumping model to file cache /tmp/jieba.cache
Loading model cost 0.653 seconds.
Prefix dict has been built successfully.
```

训练集准确率：0.936

测试集准确率：0.878

测试集评估：

	precision	recall	f1-score	support
0	0.87	0.88	0.88	592
1	0.88	0.88	0.88	608
accuracy			0.88	1200
macro avg	0.88	0.88	0.88	1200
weighted avg	0.88	0.88	0.88	1200

步骤 5 单篇文章测试

测试单篇文章预测结果：

输入：

```
nb_classifier.single_predict("外观很漂亮，出人意料地漂亮，做工非常好")
nb_classifier.single_predict("书的内容没什么好说的，主要是纸张、印刷太差，所用的纸非常粗糙比一般的盗版书还要差，裁的也不好。")
```

输出：

```
nb_classifier.single_predict("外观很漂亮，出人意料地漂亮，做工非常好")
('正面', 0.8507607980257825)
nb_classifier.single_predict("书的内容没什么好说的，主要是纸张、印刷太差，所用的纸非常粗糙比一般的盗版书还要差，裁的也不好。")
('负面', 0.7159780847706174)
```

1.3.3 SVM 文本分类

步骤 1 新建 notebook 文件，选择 pytorch 环境

步骤 2 导入相关模块

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

步骤 3 加载数据

```
train_data = pd.read_csv('data/chnsenticorp/train.tsv', sep='\t')
test_data = pd.read_csv('data/chnsenticorp/test.tsv', sep='\t')
label_map = {0: '负面', 1: '正面'}
x_train, y_train = train_data.text_a.values, train_data.label.values
x_test, y_test = test_data.text_a.values, test_data.label.values
```

步骤 4 定义分类器主类，定义训练，测试函数

```
class SVM_Classifier(object):

    def __init__(self, use_chi=False):

        self.use_chi = use_chi # 是否使用卡方检验做特征选择
        # 朴素贝叶斯分类器
        self.model = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
        # 使用 tf-idf 特征提取
        self.feature_processor = TfidfVectorizer(tokenizer=jieba.cut)
        # 卡方检验特征选择
        if use_chi:
            self.feature_selector = SelectKBest(chi2, k=10000) # 34814 -> 10000

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

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

        train_accuracy = self.model.score(x_train_fea, y_train)
        print("训练集准确率: {}".format(round(train_accuracy, 3)))

        x_test_fea = self.feature_processor.transform(x_test)
        if self.use_chi:
            x_test_fea = self.feature_selector.transform(x_test_fea)
        y_predict = self.model.predict(x_test_fea)
        test_accuracy = accuracy_score(y_test, y_predict)
        print("测试集准确率: {}".format(round(test_accuracy, 3)))
        print('测试集评估矩阵: ')
        print(classification_report(y_test, y_predict, target_names=['负面', '正面']))

    def single_predict(self, text):
        text_preprocess = [" ".join(jieba.cut(text))]
        text_fea = self.feature_processor.transform(text_preprocess)
        if self.use_chi:
            text_fea = self.feature_selector.transform(text_fea)
        predict_idx = self.model.predict(text_fea)[0]
```

```
predict_label = label_map[predict_idx]

return predict_label
```

步骤 5 训练 SVM 分类器，不使用卡方检验

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

测试集评估结果：

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

Building prefix dict from the default dictionary ...
Loading model from cache /tmp/jieba.cache
Loading model cost 0.573 seconds.
Prefix dict has been built successfully.

训练集准确率：0.968
测试集准确率：0.908
测试集评估矩阵：

	precision	recall	f1-score	support
负面	0.89	0.92	0.91	592
正面	0.92	0.89	0.91	608
accuracy			0.91	1200
macro avg	0.91	0.91	0.91	1200
weighted avg	0.91	0.91	0.91	1200

步骤 6 训练 SVM 分类器，使用卡方检验

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

测试集评估结果：

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

训练集准确率：0.948

测试集准确率：0.908

测试集评估矩阵：

	precision	recall	f1-score	support
负面	0.89	0.92	0.91	592
正面	0.92	0.89	0.91	608
accuracy			0.91	1200
macro avg	0.91	0.91	0.91	1200
weighted avg	0.91	0.91	0.91	1200

步骤 7 卡方特征分析

输入：

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

# 查看卡方值 top500 词
feature_analysis()[:500]
```

输出：

```
feature_analysis()[:500]
[('不错', 114.54110457506329),
 ('外观', 37.86973466548298),
 ('很', 35.45325438757986),
 ('配置', 33.65380068757583),
 ('性价比', 30.669976911817592),
 ('不', 28.302608532460106),
 ('差', 26.423210259544987),
 ('?', 24.93514056029271),
 ('好', 23.128495565059936),
 ('太', 22.766073651024286),
 ('没有', 21.352932367081223),
 ('性能', 19.955847037084673),
 ('很快', 19.65797398532765),
 ('速度', 19.60905866035379),
 ('漂亮', 19.094815375795335),
 ('不好', 17.92172924314892),
 ('失望', 17.8547944709826),
 ('很漂亮', 17.00132278910464),
 ('很差', 16.92606538977758),
```

步骤 8 单篇文章测试

```
svm_classifier.single_predict("外观很漂亮，出人意料地漂亮，做工非常好")
svm_classifier.single_predict("书的内容没什么好说的，主要是纸张、印刷太差，所用的纸非常粗糙比一般的盗版书还要差，裁的也不好。")
```

```
svm_classifier.single_predict("外观很漂亮，出人意料地漂亮，做工非常好")
'正面'

svm_classifier.single_predict("书的内容没什么好说的，主要是纸张、印刷太差，所用的纸非常粗糙比一般的盗版书还要差，裁的也不好。")
'负面'
```

1.3.4 TextCNN 文本分类

步骤 1 新建 notebook 文件，选择 pytorch 环境

步骤 2 导入相关库

```
import jieba
import numpy as np
import torch
import torch.nn as nn
from torch.nn.utils.rnn import pad_sequence
import pandas as pd
import torch.nn.functional as F
import torch.optim as optim
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader, TensorDataset
```

步骤 3 划分数据集

```
train_data = pd.read_csv('data/chnsenticorp/train.tsv', sep='\t')
valid_data = pd.read_csv('data/chnsenticorp/dev.tsv', sep='\t')
test_data = pd.read_csv('data/chnsenticorp/test.tsv', sep='\t')
x_train, y_train = train_data.text_a.values, train_data.label.values # 训练集
x_valid, y_valid = valid_data.text_a.values, valid_data.label.values # 验证集
x_test, y_test = test_data.text_a.values, test_data.label.values # 测试集
```

步骤 4 分词，把词加入词汇表

```
vocab = set()
cut_docs = train_data.text_a.apply(lambda x: jieba.cut(x)).values
for doc in cut_docs:
    for word in doc:
        if word.strip():
            vocab.add(word.strip())

# 将词表写入本地 vocab.txt 文件
with open('data/vocab.txt', 'w') as file:
    for word in vocab:
        file.write(word)
```

```
file.write('\n')
```

步骤 5 定义配置类

```
class Config():
    embedding_dim = 300 # 词向量维度
    max_seq_len = 200 # 文章最大词数
    vocab_file = 'data/vocab.txt' # 词汇表文件路径
    config = Config()
```

步骤 6 定义输入文本预处理的类

```
class Preprocessor:
    def __init__(self, config):
        self.config = config
        self.token2idx = {"[PAD]": 0, "[UNK]": 1}
        with open(config.vocab_file, 'r', encoding='utf-8') as reader:
            for index, line in enumerate(reader):
                token = line.strip()
                self.token2idx[token] = index + 2
        self.vocab_size = len(self.token2idx)
        self.embedding = nn.Embedding(
            num_embeddings=self.vocab_size,
            embedding_dim=self.config.embedding_dim,
            padding_idx=self.token2idx['[PAD]']
        )

    def transform(self, text_list):
        # 文本分词，并将词转换成相应的 id
        idx_list = [
            [self.token2idx.get(word.strip(), self.token2idx['[UNK]']) for word in jieba.cut(text)]
            for text in text_list
        ]

        # 将列表转换为 PyTorch 的 Tensor，并确保是 LongTensor
        idx_tensors = [torch.tensor(seq, dtype=torch.long) for seq in idx_list]

        # 填充序列到统一长度
        idx_padding = pad_sequence(
            idx_tensors,
            batch_first=True,
            padding_value=self.token2idx['[PAD]']
        )

        # 如果序列长度超过 max_seq_len，截断
        if self.config.max_seq_len:
            idx_padding = idx_padding[:, :self.config.max_seq_len]
```

```

        return idx_padding # 返回索引，而不是词向量

# 测试
preprocessor = Preprocessor(config)
result = preprocessor.transform(['我爱上海', '我爱杭州'])
print(result.shape) # 输出形状: (batch_size, max_seq_len, embedding_dim)
print(result)

```

输出：

```

torch.Size([2, 3])
tensor([[23722, 28242, 14370],
        [23722, 28242, 29230]])

```

步骤 7 设置训练芯片的类型

```

# 检测设备
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Using device: {device}')

```

步骤 8 将输入文本转换为索引，并创建数据加载器

```

# 使用 Preprocessor 将文本转换为索引
preprocessor = Preprocessor(config)
x_train_idx = preprocessor.transform(x_train).to(device) # 形状: (num_samples, max_seq_len)
x_valid_idx = preprocessor.transform(x_valid).to(device)
x_test_idx = preprocessor.transform(x_test).to(device)

# 将标签转换为 Tensor
y_train_tensor = torch.tensor(y_train, dtype=torch.long).to(device)
y_valid_tensor = torch.tensor(y_valid, dtype=torch.long).to(device)
y_test_tensor = torch.tensor(y_test, dtype=torch.long).to(device)

# 创建 DataLoader
train_dataset = TensorDataset(x_train_idx, y_train_tensor)
valid_dataset = TensorDataset(x_valid_idx, y_valid_tensor)
test_dataset = TensorDataset(x_test_idx, y_test_tensor)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=64, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

```

步骤 9 创建 TextCNN 模型

```

class TextCNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, num_classes, kernel_sizes=[3, 4, 5], num_filters=100):

```

```
super(TextCNN, self).__init__()
self.embedding = nn.Embedding(vocab_size, embedding_dim)
self.convs = nn.ModuleList([
    nn.Conv2d(1, num_filters, (k, embedding_dim)) for k in kernel_sizes
])
self.fc = nn.Linear(len(kernel_sizes) * num_filters, num_classes)
self.dropout = nn.Dropout(0.5)

def forward(self, x):
    # 输入形状: (batch_size, max_seq_len)
    x = self.embedding(x.long()) # 确保输入是 torch.long 类型
    x = x.unsqueeze(1) # 形状: (batch_size, 1, max_seq_len, embedding_dim)

    # 卷积 + ReLU + 最大池化
    conv_results = [
        F.relu(conv(x)).squeeze(3) for conv in self.convs
    ] # 每个卷积结果的形状: (batch_size, num_filters, max_seq_len - kernel_size + 1)

    pooled_results = [
        F.max_pool1d(result, result.size(2)).squeeze(2) for result in conv_results
    ] # 每个池化结果的形状: (batch_size, num_filters)

    # 拼接所有池化结果
    x = torch.cat(pooled_results, 1) # 形状: (batch_size, len(kernel_sizes) * num_filters)
    x = self.dropout(x)

    # 全连接层
    logits = self.fc(x) # 形状: (batch_size, num_classes)
    return logits
```

步骤 10 创建训练函数

```
def train(model, train_loader, valid_loader, criterion, optimizer, num_epochs=10, device=device):
    model.to(device)
    best_val_acc = 0.0

    for epoch in range(num_epochs):
        model.train()
        train_loss, train_correct, train_total = 0.0, 0, 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            # 前向传播
            outputs = model(inputs)
            loss = criterion(outputs, labels)
```

```
# 反向传播
optimizer.zero_grad()
loss.backward()
optimizer.step()

# 统计训练损失和准确率
train_loss += loss.item()
_, predicted = torch.max(outputs, 1)
train_total += labels.size(0)
train_correct += (predicted == labels).sum().item()

# 验证集评估
val_loss, val_acc = evaluate(model, valid_loader, criterion, device)

# 打印训练和验证结果
print(f'Epoch {epoch + 1}/{num_epochs}:')
print(f'Train Loss: {train_loss / len(train_loader):.4f}, Train Acc: {train_correct / train_total:.4f}')
print(f'Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}')

# 保存最佳模型
if val_acc > best_val_acc:
    best_val_acc = val_acc
    torch.save(model.state_dict(), 'best_textcnn_model.pth')

print(f'Best Validation Accuracy: {best_val_acc:.4f}')
```

步骤 11 创建模型评估函数

```
def evaluate(model, data_loader, criterion, device=device):
    model.eval()
    total_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for inputs, labels in data_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            # 前向传播
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            # 统计损失和准确率
            total_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
```

```
# 计算平均损失和准确率
avg_loss = total_loss / len(data_loader)
accuracy = correct / total
return avg_loss, accuracy
```

步骤 12 主程序，训练模型，评估模型并保存模型

```
if __name__ == '__main__':
    # 超参数
    vocab_size = preprocessor.vocab_size
    embedding_dim = config.embedding_dim
    num_classes = len(set(y_train)) # 类别数
    kernel_sizes = [3, 4, 5]
    num_filters = 100
    learning_rate = 0.001
    num_epochs = 1

    # 初始化模型
    model = TextCNN(vocab_size, embedding_dim, num_classes, kernel_sizes, num_filters)
    model.to(device)

    # 损失函数和优化器
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    # 训练和验证
    train(model, train_loader, valid_loader, criterion, optimizer, num_epochs, device=device)

    # 测试
    evaluate(model, test_loader, criterion, device=device)
```

输出：

```
Epoch 1/1:
Train Loss: 0.5938, Train Acc: 0.7006
Val Loss: 0.4185, Val Acc: 0.8283
Best Validation Accuracy: 0.8283
```

步骤 13 创建测试案例的类

```
# 加载模型
def load_model(model_path, vocab_size, embedding_dim, num_classes):
    model = TextCNN(vocab_size, embedding_dim, num_classes)
    # 使用 weights_only=True 加载模型
    model.load_state_dict(torch.load(model_path, map_location='cpu', weights_only=True)) # 加载模型参数
    model.eval() # 设置模型为评估模式
```

```
    return model
#案例测试类
# 单句预测
class TextCNNPredictor:
    def __init__(self, model_path, config):
        self.preprocessor = Preprocessor(config)
        self.model = load_model(
            model_path,
            vocab_size=self.preprocessor.vocab_size,
            embedding_dim=config.embedding_dim,
            num_classes=2 # 二分类任务
        )
        self.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self.model.to(self.device)

    def single_predict(self, text):
        # 将输入文本转换为模型输入格式
        text_idx = self.preprocessor.transform([text]).to(self.device)

        # 模型预测
        with torch.no_grad():
            outputs = self.model(text_idx)
            _, predicted = torch.max(outputs, 1)

        # 返回预测结果
        return predicted.item()
```

步骤 14 单句测试

```
# 测试单句预测，初始化预测器
model_path = 'best_textcnn_model.pth' # 训练好的模型路径
textcnn = TextCNNPredictor(model_path, config)

# 单句预测
text1 = "我不太喜欢这个变化。"
text2 = "我是很喜欢这段小说的表述的。"

print(f"文本: {text1}")
print(f"预测结果: {textcnn.single_predict(text1)}")

print(f"文本: {text2}")
print(f"预测结果: {textcnn.single_predict(text2)}")
```

输出：

```
文本：我不太喜欢这个变化。  
预测结果：0  
文本：我是很喜欢这段小说的表述的。  
预测结果：1
```

1.3.5 思考题

- 如果使用 RNN 实现文本分类效果是否会更好？

1.4 思考题

- 如果使用 RNN 实现文本分类效果是否会更好？

1.5 实验小结

本章主要介绍了自然语言处理任务中文本分类任务的具体实现，通过情感分析这样一个应用案例，比较了朴素贝叶斯、SVM 和 CNN 三种不同算法的差异，通过实验使学员加深对文本分类任务的理解，也加深对朴素贝叶斯、SVM 和 CNN 算法的理解。

2 机器翻译

2.1 实验简介

本实验通过 PyTorch 框架实现基于 Transformer 模型的机器翻译系统。通过此实验学员将深入了解 Transformer 模型的基本原理，并掌握使用 PyTorch 进行模型定义、数据预处理、训练与评估的方法。

2.2 实验环境

实验使用语言为 Python，框架为 PyTorch。

- Python3.9 及以上
- torch>=2.1 (GPU、cuda11.8/12.4)

本实验所需要的数据及代码均打包在文件 ChineseNMT-Transformer-PyTorch.zip 中。

数据集来自 WMT 2018 Chinese-English track (Only NEWS Area)
(<https://statmt.org/wmt18/translation-task.html>) 。

步骤 1 下载源码和数据并解压

```
!wget https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIP-AI%20EI%20Developer/V2.5/chapter4/ChineseNMT-Transformer-Pytorch.zip
```

输入：

```
!unzip ChineseNMT-Transformer-PyTorch.zip
```

解压文件后进入该目录执行以下操作（或者修改脚本导入路径）。

步骤 2 依赖包安装

安装所需工具库。

输入：

```
!pip install -r requirements.txt
```

2.3 实验步骤

步骤 1 导入依赖库

输入：

```
import utils
import config
import logging
import numpy as np
from tqdm import tqdm
import json

import torch
from torch.utils.data import DataLoader
from data_loader import MTDataset
from model import make_model, LabelSmoothing
from train import MultiGPULossCompute
from utils import chinese_tokenizer_load
from beam_decoder import beam_search
from model import batch_greedy_decode
import sacrebleu

import warnings
warnings.filterwarnings('ignore') # 屏蔽 WARNING 信息
```

步骤 2 词表文件生成

数据集为 en-cn 的 sentence pair，存放在 data/json 文件夹下的文件中。（说明：data/json-full 存放全量数据集，想要基于全量数据集训练的话，可以使用该路径的数据文件）

通过 BPE 算法逐步建造一个基于 subword unit 的词汇库来表示所有的词汇，该词汇库由频率出现最多的 subword 组成。本实验使用 sentencepiece 实现 BPE 分词。

方式一：

输入：

```
files = ['train', 'dev', 'test']
ch_path = './data/corpus.ch'
en_path = './data/corpus.en'
ch_lines = []
en_lines = []

for file in files:
    corpus = json.load(open('./data/json/' + file + '.json', 'r'))
    for item in corpus:
        ch_lines.append(item[1] + '\n')
        en_lines.append(item[0] + '\n')

with open(ch_path, "w") as fch:
    fch.writelines(ch_lines)

with open(en_path, "w") as fen:
    fen.writelines(en_lines)
```

```
# lines of Chinese: 252777
print("lines of Chinese: ", len(ch_lines))
# lines of English: 252777
print("lines of English: ", len(en_lines))
print("----- Get Corpus ! -----")
```

输出：

```
lines of Chinese: 6141
lines of English: 6141
----- Get Corpus ! -----
```

输入：

```
import sentencepiece as spm

def train(input_file, vocab_size, model_name, model_type, character_coverage):
    """
    search on https://github.com/google/sentencepiece/blob/master/doc/options.md to learn more
    about the parameters
    :param input_file: one-sentence-per-line raw corpus file. No need to run tokenizer, normalizer or
    preprocessor.
                        By default, SentencePiece normalizes the input with Unicode NFKC.
                        You can pass a comma-separated list of files.
    :param vocab_size: vocabulary size, e.g., 8000, 16000, or 32000
    :param model_name: output model name prefix. <model_name>.model and <model_name>.vocab
    are generated.
    :param model_type: model type. Choose from unigram (default), bpe, char, or word.
                        The input sentence must be pretokenized when using word type.
    :param character_coverage: amount of characters covered by the model, good defaults are: 0.9995
    for languages with
                                rich character set like Japanese or Chinese and 1.0 for other languages
    with
                                small character set.
    """
    input_argument = '--input=%s --model_prefix=%s --vocab_size=%s --model_type=%s --
    character_coverage=%s ' \
                    '--pad_id=0 --unk_id=1 --bos_id=2 --eos_id=3 '
    cmd = input_argument % (input_file, model_name, vocab_size, model_type, character_coverage)
    spm.SentencePieceTrainer.Train(cmd)

def piece_run():
    en_input = './data/corpus.en' # 上述生成的文本文件
    en_vocab_size = 26889 # 32000
    en_model_name = './tokenizer/eng'
    en_model_type = 'bpe'
    en_character_coverage = 1
    train(en_input, en_vocab_size, en_model_name, en_model_type, en_character_coverage)
```

```
ch_input = './data/corpus.ch'
ch_vocab_size = 26889 # 32000
ch_model_name = './tokenizer/chn'
ch_model_type = 'bpe'
ch_character_coverage = 0.9995
train(ch_input, ch_vocab_size, ch_model_name, ch_model_type, ch_character_coverage)

piece_run()
```

输出：

```
----- english finished! -----
----- chinese finished! -----
```

此时会在当前路径下生成分词模型 chn.model 和 eng.model、词表 chn.vocab 和 eng.vocab。

方法二：

如果上述方法一时间太长，直接解压 tokenizer 下的 token.zip 文件即可。

输入：

```
!unzip -d ./tokenizer ./tokenizer/token.zip
```

输出：

```
Archive: ./tokenizer/token.zip
  inflating: ./tokenizer/eng.model
  inflating: ./tokenizer/eng.vocab
  inflating: ./tokenizer/chn.model
  inflating: ./tokenizer/chn.vocab
```

如果想查看分词结果，可以继续运行如下代码。

输入：

```
def piece_test():
    sp = spm.SentencePieceProcessor()
    text = "美国总统特朗普今日抵达夏威夷。"

    sp.Load("./chn.model")
    print(sp.EncodeAsPieces(text))
    print(sp.EncodeAsIds(text))
    a = [12907, 277, 7419, 7318, 18384]
    print(sp.decode_ids(a))

piece_test()
```

输出：

```
['_美国', '总统', '特朗普', '今日', '抵达', '夏威夷', '。']
[352, 78, 213, 6918, 10400, 12176, 24137]
的定义历史民事掀起洛夫现在
```

可以看到：sentencepiece 将每句文本的开头和空格均用 “_” 表示。

步骤 3 查看 GPU 是否可用并设置运行环境

将运行环境设置为 GPU。

输入：

```
print("GPU 数量: ",torch.cuda.device_count())
print("GPU 是否可用: ",torch.cuda.is_available())

import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0'
```

输出：

```
GPU 数量: 1
GPU 是否可用: True
```

步骤 4 设置超参数和常量

config.py 中定义了所用到的超参数和常量。

包括：epoch 数量、batch size、数据存放路径、权重文件存放路径等。

```
import torch

# 模型维度
d_model = 512 # 输入嵌入的维度
n_heads = 8   # 多头注意力的头数
n_layers = 6  # 编码器和解码器的层数
d_k = 64      # 每个头的维度
d_v = 64      # 每个头的值维度（通常与 d_k 相同）
d_ff = 2048   # 前馈网络的维度
dropout = 0.1 # Dropout 比率

# 特殊标记索引
padding_idx = 0 # 填充标记的索引
bos_idx = 2     # 开始序列标记的索引
eos_idx = 3     # 结束序列标记的索引

# 词汇表大小
src_vocab_size = 32000 # 源语言词汇表大小
tgt_vocab_size = 32000 # 目标语言词汇表大小

# 训练参数
batch_size = 32      # 每个批次的大小
epoch_num = 2        # 训练的总轮数，此处做展示设置为 2，如果时间充裕可以增大该值
early_stop = 5       # 提前停止训练的轮数（如果性能不再提升）
lr = 3e-4            # 学习率
```

```
# 解码参数
max_len = 60          # 贪心解码时的最大句子长度
beam_size = 3         # 用于 BLEU 评分的 beam search 大小

# 其他选项
use_smoothing = False # 是否使用标签平滑
use_noamopt = True    # 是否使用 Noam 学习率调度器

# 数据和模型路径
data_dir = './data'
train_data_path = './data/json/train.json'
dev_data_path = './data/json/dev.json'
test_data_path = './data/json/test.json'
model_path = './experiment/model.pth'      # 模型保存路径
pretrain_model_path = './pretrain/model.pth' # 预训练模型路径
log_path = './experiment/train.log'        # 训练日志路径
output_path = './experiment/output.txt'    # 输出结果路径

# GPU 和设备设置
gpu_id = ""           # 要使用的 GPU 的 ID（空字符串表示不使用 GPU）
device_id = [0]       # 设备 ID 列表（用于多 GPU 训练，这里仅作为示例）

# 设置设备
device = torch.device('cpu')
```

步骤 5 获取数据

data_loader.py 模块中定义了数据处理相关操作。主要是基于 torch.utils.data.Dataset 进行数据的预处理，主要包括：

- （1）把原始语料中的中英文句对按照英文句子的长度排序，使得每个 batch 中的句子长度相近。
- （2）利用训练好的分词模型分别对中英文句子进行分词，利用词表将其转换为 id。
- （3）在每个 id sequence 的首尾加上起始符和终止符，并将其转换为 Tensor。
- （4）将上述 Tensor 输入到定义的 Batch 类中，转换为 transformer 支持的输入格式。

Data_loader 文件（该文件代码已提供，不需要手写）：

```
import torch
import json
import numpy as np
from torch.autograd import Variable # 用于自动求导的变量封装
from torch.utils.data import Dataset # 提供数据集的抽象类
from torch.nn.utils.rnn import pad_sequence # 对序列进行填充，使它们具有相同的长度
from utils import english_tokenizer_load, chinese_tokenizer_load # 加载中英文分词器的工具函数
```

```
import config # 导入配置模块
DEVICE = config.device

# 生成后续位置掩码的函数
def subsequent_mask(size):
    """生成一个掩码，用于遮挡序列中的后续位置。"""
    attn_shape = (1, size, size) # 掩码的形状
    subsequent_mask = np.triu(np.ones(attn_shape), k=1).astype('uint8') # 生成一个右上角(不含主对角线)为全 1，左下角(含主对角线)为全 0 的 subsequent_mask 矩阵
    return torch.from_numpy(subsequent_mask) == 0 # 转换为 False (1 变为 0) 和 True (0 变为 1) 的掩码

# 用于存储一批训练数据的类
class Batch:
    def __init__(self, src_text, trg_text, src, trg=None, pad=0):
        """
        初始化一批数据。
        src_text: 源语言文本列表。
        trg_text: 目标语言文本列表。
        src: 源语言输入张量。
        trg: 目标语言输入张量（可为空，用于解码器输入）。
        pad: 填充标记的索引。
        """
        self.src_text = src_text # 源语言文本
        self.trg_text = trg_text # 目标语言文本
        self.src = src.to(DEVICE) # 源语言输入张量，转移到指定设备
        self.src_mask = (src != pad).unsqueeze(-2) # 源语言输入掩码
        if trg is not None:
            self.trg = trg[:, :-1].to(DEVICE) # 目标语言输入（去除最后一个标记，用于解码器）
            self.trg_y = trg[:, 1:].to(DEVICE) # 目标语言输出（去除第一个标记，用于训练目标）
            self.trg_mask = Batch.make_std_mask(self.trg, pad) # 目标语言输入掩码
            self.ntokens = (self.trg_y != pad).data.sum() # 统计非填充标记的数量

    @staticmethod
    def make_std_mask(tgt, pad):
        """创建一个掩码，用于隐藏填充标记和未来的单词。"""
        tgt_mask = (tgt != pad).unsqueeze(-2) # 目标语言输入掩码（去除填充标记）
        tgt_mask = tgt_mask & Variable(subsequent_mask(tgt.size(-1)).type_as(tgt_mask.data)) # 结合后续位置掩码
        return tgt_mask

# 自定义数据集类
class MTDataset(Dataset):
    def __init__(self, data_path):
        """
```

```

        初始化数据集。
        data_path: 数据文件的路径。
        """
        self.out_en_sent, self.out_cn_sent = self.get_dataset(data_path, sort=True) # 加载并排序中英文
句子
        self.sp_eng = english_tokenizer_load() # 加载英文分词器
        self.sp_chn = chinese_tokenizer_load() # 加载中文分词器
        self.PAD = self.sp_eng.pad_id() # 填充标记的索引
        self.BOS = self.sp_eng.bos_id() # 开始序列标记的索引
        self.EOS = self.sp_eng.eos_id() # 结束序列标记的索引

    @staticmethod
    def len_argsort(seq):
        """根据序列长度对索引进行排序。"""
        return sorted(range(len(seq)), key=lambda x: len(seq[x]))

    def get_dataset(self, data_path, sort=False):
        """
        加载数据集。
        data_path: 数据文件的路径。
        sort: 是否根据英文句子长度进行排序。
        """
        dataset = json.load(open(data_path, 'r')) # 加载数据
        out_en_sent = [] # 英文句子列表
        out_cn_sent = [] # 中文句子列表
        for idx, _ in enumerate(dataset):
            out_en_sent.append(dataset[idx][0]) # 英文句子
            out_cn_sent.append(dataset[idx][1]) # 中文句子
        if sort:
            sorted_index = self.len_argsort(out_en_sent) # 根据英文句子长度排序
            out_en_sent = [out_en_sent[i] for i in sorted_index] # 重新排序英文句子
            out_cn_sent = [out_cn_sent[i] for i in sorted_index] # 重新排序中文句子（保持与英文句子
顺序一致）
        return out_en_sent, out_cn_sent

    def __getitem__(self, idx):
        """获取指定索引处的数据项。"""
        eng_text = self.out_en_sent[idx]
        chn_text = self.out_cn_sent[idx]
        return [eng_text, chn_text]

    def __len__(self):
        """返回数据集中的数据项数量。"""
        return len(self.out_en_sent)

```

```
def collate_fn(self, batch):
    """
    自定义的 batch 处理函数。
    batch: 一批数据项。
    """
    src_text = [x[0] for x in batch]
    tgt_text = [x[1] for x in batch]

    src_tokens = [[self.BOS] + self.sp_eng.EncodeAsIds(sent) + [self.EOS] for sent in src_text] # 英文分词并添加 BOS 和 EOS
    tgt_tokens = [[self.BOS] + self.sp_chn.EncodeAsIds(sent) + [self.EOS] for sent in tgt_text] # 中文分词并添加 BOS 和 EOS

    batch_input = pad_sequence([torch.LongTensor(np.array(l_)) for l_ in src_tokens],
                               batch_first=True, padding_value=self.PAD) # 对英文输入进行填充
    batch_target = pad_sequence([torch.LongTensor(np.array(l_)) for l_ in tgt_tokens],
                                batch_first=True, padding_value=self.PAD) # 对中文目标进行填充

    return Batch(src_text, tgt_text, batch_input, batch_target, self.PAD) # 返回一个 Batch 对象
```

通过 data_loader 中的数据类创建数据集。

输入：

```
train_dataset = MTDataset(config.train_data_path)
dev_dataset = MTDataset(config.dev_data_path)
test_dataset = MTDataset(config.test_data_path)

# logging.info("----- Dataset Build! -----")
print("----- Dataset Build! -----")
train_dataloader = DataLoader(train_dataset, shuffle=True, batch_size=config.batch_size,
                              collate_fn=train_dataset.collate_fn)
dev_dataloader = DataLoader(dev_dataset, shuffle=False, batch_size=config.batch_size,
                            collate_fn=dev_dataset.collate_fn)
test_dataloader = DataLoader(test_dataset, shuffle=False, batch_size=config.batch_size,
                             collate_fn=test_dataset.collate_fn)
```

输出：

```
----- Dataset Build! -----
```

步骤 6 定义学习率

Warm up 是在 ResNet 中提到的一种针对包括 Adam 和 RMSProp 在内的一些自适应优化器的学习率预热方法。18 年，Facebook 又针对上述的 constant warmup 方法进行了改进，即从最开始的小学习率开始，每个 iteration 增大一点，直到最初设置的比较大的学习率。transformer 在 Warm Up 阶段采用的策略与 Facebook 近似。在 Warm Up 阶段之后，transformer 采用了指数的学习率递减策略。这一整体的学习率更新策略称为 NoamOpt。

输入：

```
class NoamOpt:
    """Optim wrapper that implements rate."""

    def __init__(self, model_size, factor, warmup, optimizer):
        self.optimizer = optimizer
        self._step = 0
        self.warmup = warmup
        self.factor = factor
        self.model_size = model_size
        self._rate = 0

    def step(self):
        """Update parameters and rate"""
        self._step += 1
        rate = self.rate()
        for p in self.optimizer.param_groups:
            p['lr'] = rate
        self._rate = rate
        self.optimizer.step()

    def rate(self, step=None):
        """Implement `lr` above"""
        if step is None:
            step = self._step
        return self.factor * (self.model_size ** (-0.5) * min(step ** (-0.5), step * self.warmup ** (-1.5)))

def get_std_opt(model):
    """for batch_size 32, 5530 steps for one epoch, 2 epoch for warm-up"""
    return NoamOpt(model.src_embed[0].d_model, 1, 10000,
                    torch.optim.Adam(model.parameters(), lr=0, betas=(0.9, 0.98), eps=1e-9))
```

步骤 7 模型初始化

model.py 模块中定义了 transformer 模型结构（模块中实现了 transformer 的 decoder 和 encoder 部分，并基于此实现了完成的 transformer，此部分可以直接导入，不要求手动实现）。

此步骤中加载了预训练权重，预训练权重文件存放在 pretrain 文件夹中。

输入：

```
# 初始化模型
model = make_model(config.src_vocab_size, config.tgt_vocab_size, config.n_layers,
                   config.d_model, config.d_ff, config.n_heads, config.dropout)
model.load_state_dict(torch.load(config.pretrain_model_path))
model_par = torch.nn.DataParallel(model)
```

步骤 8 定义损失函数和优化器

Transformer 中借鉴了标签平滑 (label smoothing) 的思想, 即通过在标签 y 中加入噪声, 实现对模型约束, 降低模型过拟合程度。本实验中实际上未采用标签平滑操作。

输入:

```
# 损失函数和优化器
if config.use_smoothing:
    criterion = LabelSmoothing(size=config.tgt_vocab_size, padding_idx=config.padding_idx,
                                smoothing=0.1)
    criterion.cuda()
else:
    criterion = torch.nn.CrossEntropyLoss(ignore_index=0, reduction='sum')
if config.use_noamopt:
    optimizer = get_std_opt(model)
else:
    optimizer = torch.optim.AdamW(model.parameters(), lr=config.lr)
```

步骤 9 定义训练和验证过程函数

输入:

```
def run_epoch(data, model, loss_compute):
    total_tokens = 0.
    total_loss = 0.

    for batch in tqdm(data):
        out = model(batch.src, batch.trg, batch.src_mask, batch.trg_mask)
        loss = loss_compute(out, batch.trg_y, batch.ntokens)

        total_loss += loss
        total_tokens += batch.ntokens
    return total_loss / total_tokens

def train(train_data, dev_data, model, model_par, criterion, optimizer):
    """训练并保存模型"""
    # 初始化模型在 dev 集上的最优 Loss 为一个较大值
    best_bleu_score = 0.0
    early_stop = config.early_stop
    for epoch in range(1, config.epoch_num + 1):
        # 模型训练
        model.train()
        train_loss = run_epoch(train_data, model_par,
                                MultiGPULossCompute(model.generator, criterion, config.device_id,
                                                        optimizer))
        # logging.info("Epoch: {}, loss: {}".format(epoch, train_loss))
        print("Epoch: {}, loss: {}".format(epoch, train_loss))
        # 模型验证
        model.eval()
```

```

dev_loss = run_epoch(dev_data, model_par,
                      MultiGPULossCompute(model.generator, criterion, config.device_id,
None))
bleu_score = evaluate(dev_data, model)
# logging.info('Epoch: {}, Dev loss: {}, Bleu Score: {}'.format(epoch, dev_loss, bleu_score))
print('Epoch: {}, Dev loss: {}, Bleu Score: {}'.format(epoch, dev_loss, bleu_score))

# 如果当前 epoch 的模型在 dev 集上的 loss 优于之前记录的最优 loss 则保存当前模型，并更新最优
loss 值
if bleu_score > best_bleu_score:
    torch.save(model.state_dict(), config.model_path)
    best_bleu_score = bleu_score
    early_stop = config.early_stop
    # logging.info("----- Save Best Model! -----")
    print("----- Save Best Model! -----")
else:
    early_stop -= 1
    logging.info("Early Stop Left: {}".format(early_stop))
if early_stop == 0:
    # logging.info("----- Early Stop! -----")
    print("----- Early Stop! -----")
    break

def evaluate(data, model, mode='dev', use_beam=True):
    """在 data 上用训练好的模型进行预测，打印模型翻译结果"""
    sp_chn = chinese_tokenizer_load()
    trg = []
    res = []
    with torch.no_grad():
        # 在 data 的英文数据长度上遍历下标
        for batch in tqdm(data):
            # 对应的中文句子
            cn_sent = batch.trg_text
            src = batch.src
            src_mask = (src != 0).unsqueeze(-2)
            if use_beam:
                decode_result, _ = beam_search(model, src, src_mask, config.max_len,
                                                config.padding_idx, config.bos_idx, config.eos_idx,
                                                config.beam_size, config.device)
            else:
                decode_result = batch_greedy_decode(model, src, src_mask,
                                                    max_len=config.max_len)
            decode_result = [h[0] for h in decode_result]
            translation = [sp_chn.decode_ids(_s) for _s in decode_result]
            trg.extend(cn_sent)
            res.extend(translation)
    if mode == 'test':

```

```

with open(config.output_path, "w") as fp:
    for i in range(len(trg)):
        line = "idx:" + str(i) + trg[i] + '|||' + res[i] + '\n'
        fp.write(line)
trg = [trg]
bleu = sacrebleu.corpus_bleu(res, trg, tokenize='zh')
return float(bleu.score)

```

步骤 10 训练

为减少训练时间，本实验中实际大量减少了参与训练的数据量，但压缩文件中包含了完整数据，可自行使用。另外，本实验设置的 epoch=2，此参数可以再 config 文件中修改。本次 2 个 epoch 花费大概 20 分钟的时间。训练过程中产生的日志信息、权重文件等保存在 experiment 文件夹中。

输入：

```

print("----- Get Dataloader! -----")
train(train_dataloader, dev_dataloader, model, model_par, criterion, optimizer)

```

输出：

```

----- Get Dataloader! -----
100%|██████████████████| 63/63 [00:16<00:00, 3.78it/s]
Epoch: 1, loss: 12.050097465515137
100%|██████████████████| 85/85 [00:04<00:00, 17.41it/s]
100%|██████████████████| 85/85 [05:37<00:00, 3.98s/it]
Epoch: 1, Dev loss: 12.136094093322754, Bleu Score: 0.2165632839994105
----- Save Best Model! -----
100%|██████████████████| 63/63 [00:17<00:00, 3.69it/s]
Epoch: 2, loss: 11.580182075500488
100%|██████████████████| 85/85 [00:05<00:00, 16.71it/s]
100%|██████████████████| 85/85 [05:32<00:00, 3.91s/it]
Epoch: 2, Dev loss: 11.472822189331055, Bleu Score: 0.1997061794353996

```

步骤 11 测试

使用测试数据集完成模型效果测试。

输入：

```

def test(data, model, criterion):
    with torch.no_grad():
        # 加载模型
        model.load_state_dict(torch.load(config.model_path))
        model_par = torch.nn.DataParallel(model)
        model.eval()
        # 开始预测
        test_loss = run_epoch(data, model_par,
                               MultiGPULossCompute(model.generator, criterion, config.device_id,
None))
        bleu_score = evaluate(data, model, 'test')

```

```
# logging.info('Test loss: {}, Bleu Score: {}'.format(test_loss, bleu_score))
print('Test loss: {}, Bleu Score: {}'.format(test_loss, bleu_score))
```

```
test(test_dataloader, model, criterion)
```

输出：

```
100%|████████████████████| 45/45 [00:02<00:00, 15.87it/s]
100%|████████████████████| 45/45 [03:04<00:00, 4.10s/it]
Test loss: 12.141939163208008, Bleu Score: 0.18536807099398378
```

步骤 12 推理

前面步骤已经实现了模型的训练、验证和推理。由于训练样本经过一定的处理体量变小，因为词典大小受限。为保证推理效果，后续推理过程，我们仍然使用全量的数据集所产生的词典进行推理。

输入：

```
from utils import english_tokenizer_load_inf
from utils import chinese_tokenizer_load_inf

def translate(src, model, use_beam=True):
    """用训练好的模型进行预测单句，打印模型翻译结果"""
    sp_chn = chinese_tokenizer_load_inf()
    with torch.no_grad():
        model.load_state_dict(torch.load(config.model_path))
        model.eval()
        src_mask = (src != 0).unsqueeze(-2)
        if use_beam:
            decode_result, _ = beam_search(model, src, src_mask, config.max_len,
                                           config.padding_idx, config.bos_idx, config.eos_idx,
                                           config.beam_size, config.device)
            decode_result = [h[0] for h in decode_result]
        else:
            decode_result = batch_greedy_decode(model, src, src_mask, max_len=config.max_len)
        translation = [sp_chn.decode_ids(_s) for _s in decode_result]
        print(translation[0])

def one_sentence_translate(sent, beam_search=True):
    # 初始化模型
    model = make_model(config.src_vocab_size, config.tgt_vocab_size, config.n_layers,
                      config.d_model, config.d_ff, config.n_heads, config.dropout)
    BOS = english_tokenizer_load_inf().bos_id() # 2
    EOS = english_tokenizer_load_inf().eos_id() # 3
    src_tokens = [[BOS] + english_tokenizer_load_inf().EncodeAsIds(sent) + [EOS]]
    batch_input = torch.LongTensor(np.array(src_tokens)).to(config.device)
```

```
translate(batch_input, model, use_beam=beam_search)

def translate_example():
    """单句翻译示例"""
    sent = "The near-term policy remedies are clear: raise the minimum wage to a level that will keep a" \
          "fully employed worker and his or her family out of poverty, and extend the earned-income" \
          "tax credit " \
          "to childless workers."
    # tgt: 近期的政策对策很明确：把最低工资提升到足以一个全职工人及其家庭免于贫困的水平，扩大对
    # 无子女劳动者的工资所得税减免。
    one_sentence_translate(sent, beam_search=True)

translate_example()
```

输出：

短期政策方案很清楚:把最低工资提高到充分就业的水平,并扩大向无薪工人发放所得的税收信用。

可以看到，对于输入句子 “The near-term policy remedies are clear: raise the minimum wage to a level that will keep a fully employed worker and his or her family out of poverty, and extend the earned-income tax credit to childless workers.” 的翻译结果相对比较准确的。

2.4 实验小结

本节实验使用 PyTorch 框架实现了基于 Transformer 的中英文翻译模型，通过实验使学员了解 PyTorch 的基本用法，同时加深对 Transformer 的基本原理和网络结构的理解。

2.5 思考题

请思考本实验中训练时加载的权重文件和推理时加载的权重文件是否一样？

答案：不一样。推理时加载的权重文件来自于训练阶段保存的结果。

华为认证 AI 系列教程

HCIP-AI-EI Developer

华为企业 AI 开发高级工程师认证

ModelArts

实验指导手册

版本：2.5



华为技术有限公司

版权所有 © 华为技术有限公司 2026。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<https://e.huawei.com>

华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT基础设施认证、基础软硬件认证、云平台及云服务认证三类认证。

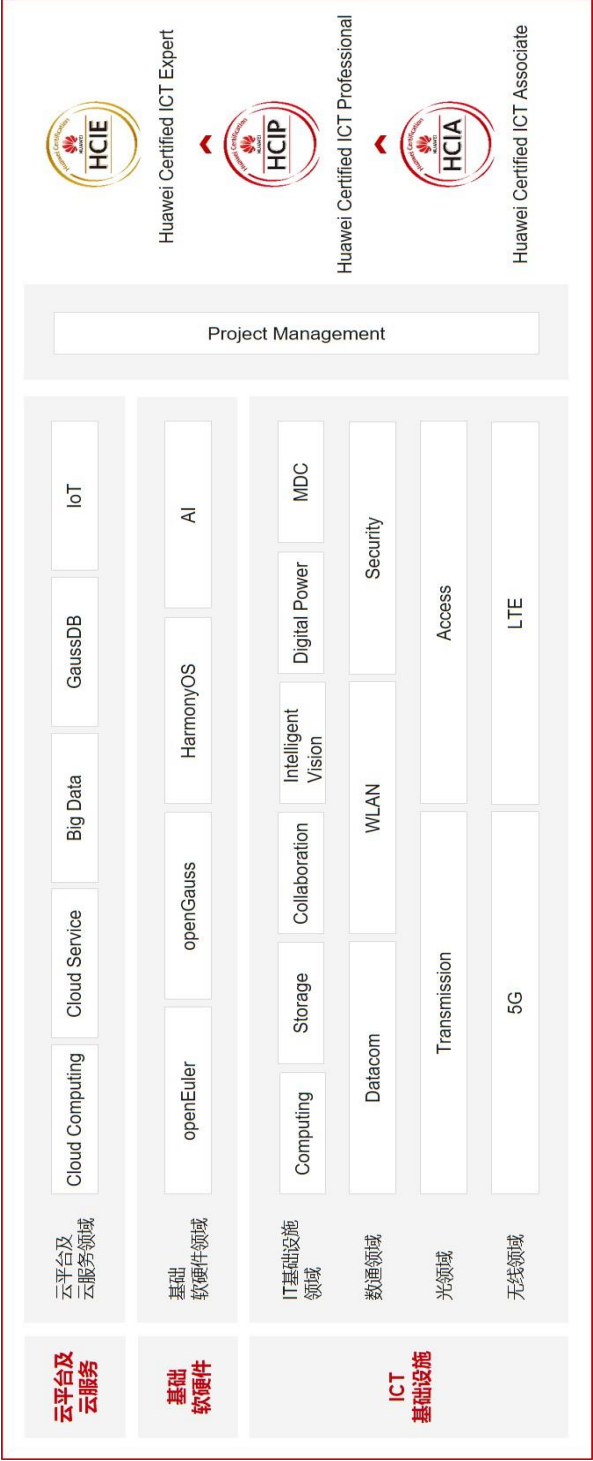
根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIP-AI-EI Developer V2.5定位于培养与认证具有图像处理、语音处理、自然语言处理能力，能够应用通用开源框架和华为云ModelArts平台等进行开发和创新的高级工程师。

通过HCIP-AI-EI Developer V2.5认证，您将掌握神经网络基础、图像处理理论、语音处理理论、自然语言处理理论、大模型深度学习算法等基础知识，具备运用通用开源框架和华为云ModelArts平台等进行图像处理、语音处理和自然语言处理等人工智能产品设计开发的能力，能够胜任AI开发高级工程师、图像处理开发工程师、语音处理开发工程师、自然语言处理开发工程师等岗位。

华为认证协助您打开行业之窗，开启改变之门，屹立在人工智能世界的潮头浪尖！



简介

本书为 HCIP-AI 认证培训教程，适用于准备参加 HCIP-AI 考试的学员或者希望了解 AI 基础知识的读者。掌握本实验手册内容，您将能够了解 AI 开发全流程，以及如何使用 ModelArts 开发 AI 应用，包括数据上传、数据标注、深度学习算法开发、模型训练、模型部署和推理等。数据集管理提供数据标注、特征分析和数据集版本管理功能，预置算法可以降低 AI 初学者的使用门槛。ModelArts 提供丰富的 AI 开发工具，是一站式的 AI 开发平台。

内容描述

本实验指导书共包含 2 组（共 6 个）实验，涉及到基于预置算法的图像分类和物体检测等算法，掌握构建 AI 应用的实践能力。

- 实验 1 数据上传 OBS
- 实验 2 创建 ModelArts 数据集
- 实验 3 数据集智能标注
- 实验 4 数据特征分析
- 实验 5 预置算法垃圾分类应用
- 实验 6 预置算法口罩检测应用

读者知识背景

本课程为华为认证开发课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 有一定的数据结构基础
- 有一定的 Python 编程基础
- 有基本的深度学习理论概念

实验环境说明

ModelArts 提供云上的开发环境（华北-北京四），无需自己安装环境。

实验数据说明

实验数据：<https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIP-AI%20EI%20Developer/V2.5/chapter6/ModelArts%E5%AE%9E%E9%AA%8C.rar>。

目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
实验环境说明	3
实验数据说明	4
1 数据集管理	6
1.1 实验简介	6
1.2 实验目的	6
1.3 实验环境说明	6
1.4 实验步骤	7
1.4.1 数据上传 OBS	7
1.4.2 创建 ModelArts 数据集	9
1.4.3 数据集智能标注	11
1.4.4 数据特征分析	16
1.5 实验小结	19
1.6 思考题	19
2 深度学习预置算法	20
2.1 实验简介	20
2.2 实验目的	20
2.3 实验步骤	20
2.3.1 垃圾分类应用	20
2.3.2 口罩检测应用	27
2.4 实验小结	34
2.5 思考题	34

1 数据集管理

1.1 实验简介

本案例将详细介绍如何利用 ModelArts 的数据管理功能，对一份数据集进行在线智能标注和数据集统计特征分析。

1.2 实验目的

- 掌握将本地图片和标注文件上传到 ModelArts 数据管理的方法
- 掌握在 ModelArts 中进行智能标注、特征分析等操作的方法

1.3 实验环境说明

如果是第一次使用 ModelArts 服务，在使用之前需要给服务添加访问授权，授权作业能够访问华为云存储 OBS，若没有添加，则无法创建作业。具体操作步骤如下：

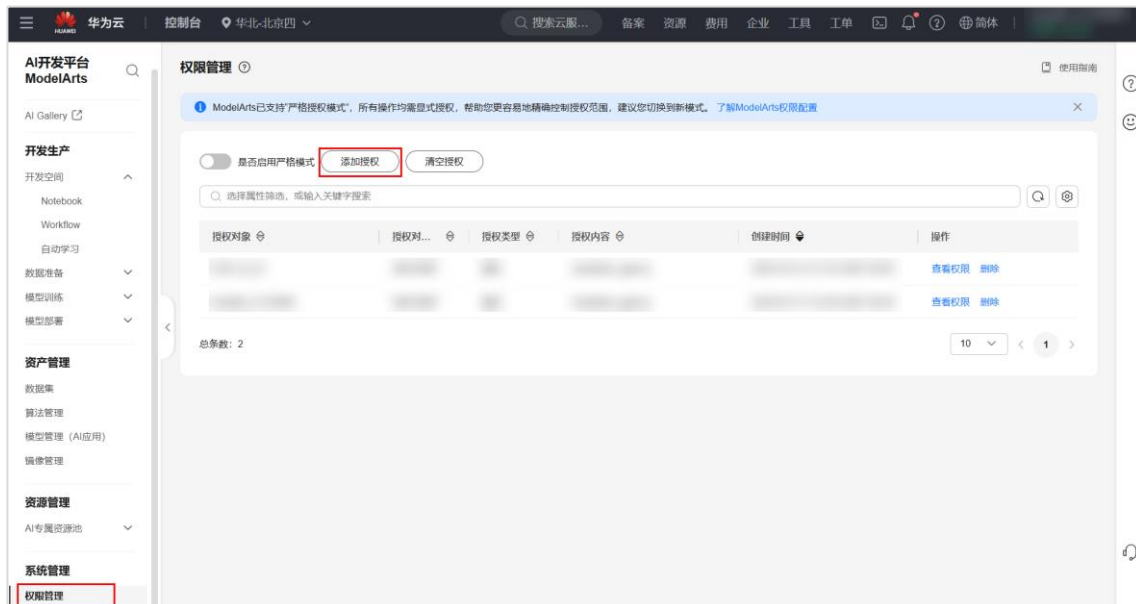


图1-1 ModelArts 操作台

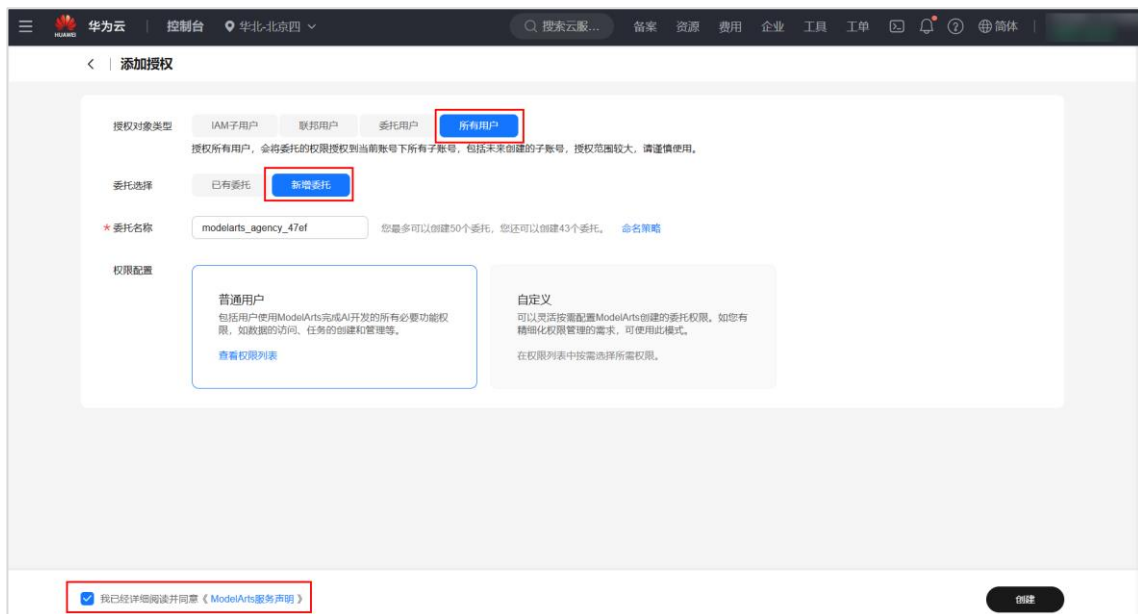


图1-2 添加访问授权

1.4 实验步骤

1.4.1 数据上传 OBS

步骤 1 创建 OBS 桶

创建数据集时，数据通过 OBS 存储。

进入控制台，将光标移动至左边栏，弹出菜单中选择“服务列表”>“存储”>“对象存储服务 OBS”，如下图所示：



图1-3 对象存储服务 OBS

点击“创建桶”按钮进入创建界面。



图1-4 创建 OBS 桶

开始创建。配置参数如下：

- 复制桶配置：不选
- 区域：华北-北京四
- 桶名称：自定义，将在后续步骤使用，下文以桶名为 modelarts-exp 为例，实际可以按需要修改
- 数据冗余存储策略：单 AZ 存储
- 默认存储类别：标准存储
- 桶策略：私有

单击“立即创建”>“确定”，完成桶创建。

OBS 桶创建完成后，点击创建的“桶名称”>“对象”>“新建文件夹”，创建 dataset-tag 文件夹，在该文件夹下继续创建三个文件夹，分明命名 input，output，result，用于上传数据输入输出的存储路径。



图1-5 OBS 桶创建文件夹

步骤 2 上传数据

本实验准备了一份只标注了部分图片的菜品检测数据集，数据集包含多种菜品图片，图片总数 100 张，且包含 10% 的图片标注信息。

下载解压完菜品数据之后，点击进入上步桶中创建的 input 文件夹。此文件夹为原始数据的存储路径，将存储包含图片和部分已标注的标签文件。

点击“上传对象”，选择本地解压路径下的 100 个菜品数据文件（Ctrl + A 可全选），点击“上传对象”，选择添加文件，如下图所示：

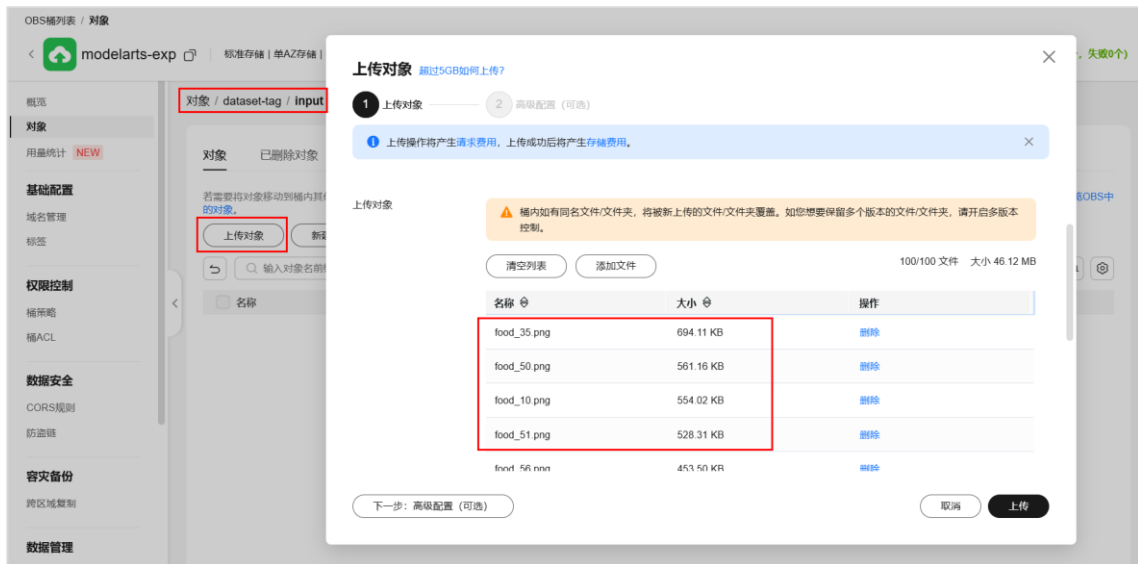


图1-6 上传数据

点击右上方任务中心，可点击查看上传任务进度。

1.4.2 创建 ModelArts 数据集

步骤 1 进入 ModelArts 数据集管理

点击控制台页面，点击左上角服务列表按钮，下拉找到【人工智能】，再找到【AI 开发平台 ModelArts】，点击进入“ModelArts”控制台页面，如下图所示：

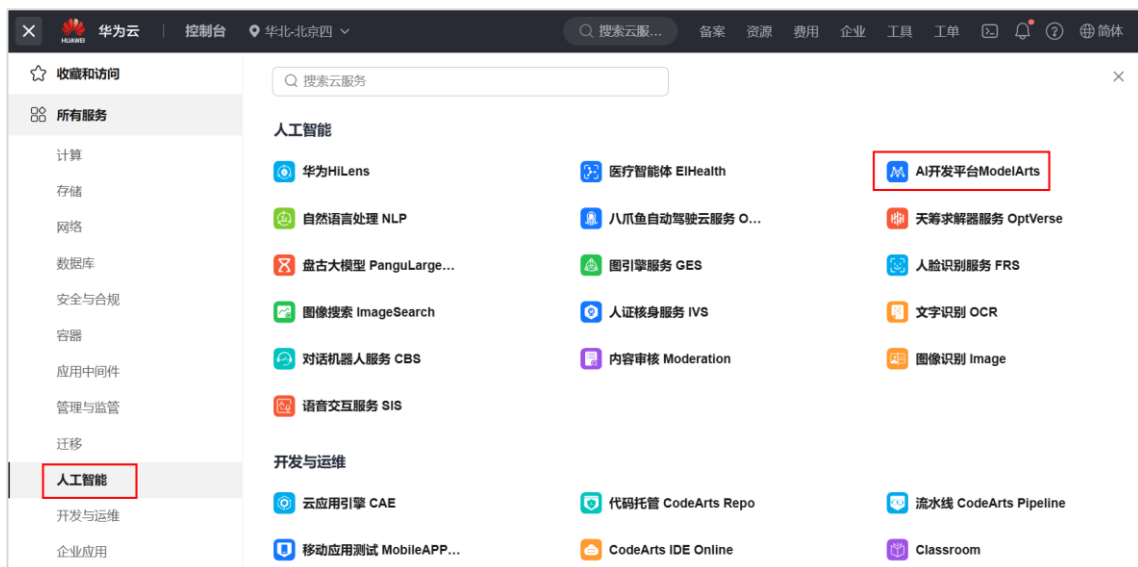


图1-7 AI 开发平台 ModelArts

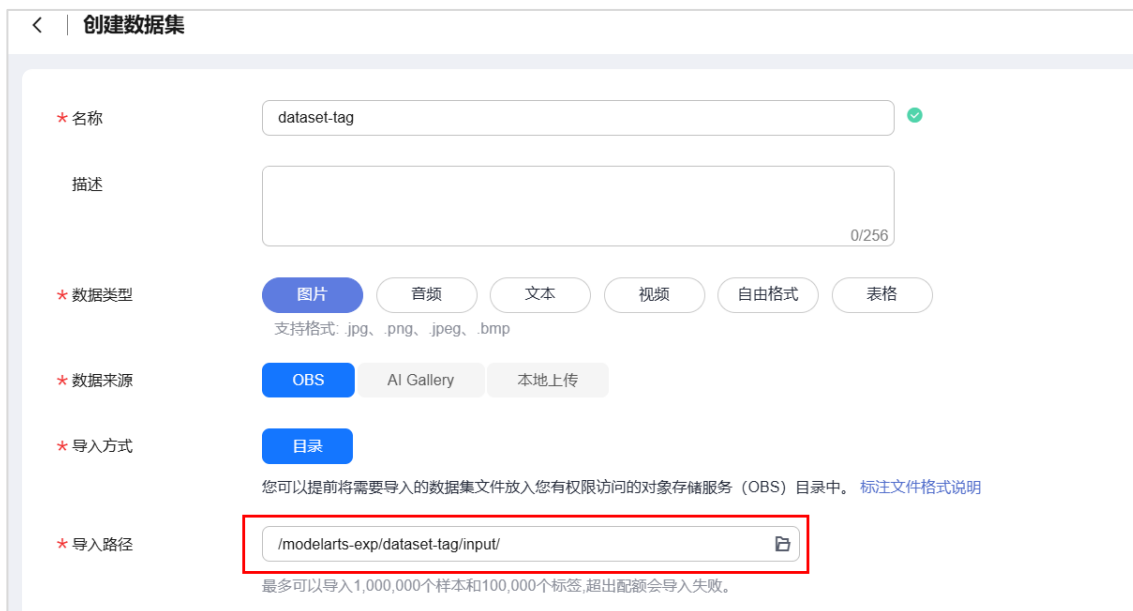
进入 ModelArts，点击资产管理 > 数据集，点击“创建数据集”，进入数据集创建页面。



图1-8 创建数据集

步骤 2 创建数据集

数据集创建页面，如下图所示：



The screenshot shows the '创建数据集' (Create Dataset) form. The form has the following fields and options:

- 名称 (Name):** dataset-tag (with a green checkmark icon).
- 描述 (Description):** A text area with a character count of 0/256.
- 数据类型 (Data Type):** Radio buttons for 图片 (Image), 音频 (Audio), 文本 (Text), 视频 (Video), 自由格式 (Free Format), and 表格 (Table). Below the buttons, it says '支持格式: jpg、png、jpeg、bmp' (Supported formats: jpg, png, jpeg, bmp).
- 数据来源 (Data Source):** Radio buttons for OBS, AI Gallery, and 本地上传 (Local Upload).
- 导入方式 (Import Method):** A radio button for 目录 (Directory).
- 导入路径 (Import Path):** A text field containing '/modelarts-exp/dataset-tag/input/' (highlighted with a red box) and a folder icon.

Below the form, there is a note: '您可以提前将需要导入的数据集文件放入您有权限访问的对象存储服务（OBS）目录中。' (You can upload the dataset files to the Object Storage Service (OBS) directory you have permission to access in advance.) followed by a link '标注文件格式说明' (Labeling File Format Specification). At the bottom, it says: '最多可以导入1,000,000个样本和100,000个标签,超出配额会导入失败。' (You can import at most 1,000,000 samples and 100,000 labels. Exceeding the quota will result in import failure.)



* 数据标注状态

未标注 **已标注**

* 标注格式

物体检测 ModelArts PASCAL VOC...

文件存放方式要求将标注对象和标注文件存储在同一目录，并且名称一一对应。标注文件需要满足PASCAL_VOC文件格式。

文件存放方式示意图

```

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

```

高级特征选项

按标签导入 ☒

* 数据集输出位置

/modelarts-exp/dataset-tag/output/

选择数据集输出位置，此位置会存放输出的标注信息等文件。此位置不能和导入路径相同且不能为导入路径的子目录。

图1-9 数据集创建页面

配置参数如下：

- 名称：自定义，便于区分
 - 描述：自定义
 - 数据类型：图片
 - 数据来源：OBS
 - 导入方式：目录
 - 导入路径：选择 1.4.1 中创建的对象存储服务（OBS）路径中的 input 目录
 - 数据标注状态：已标注
 - 标注格式：默认
 - 数据集输出位置：选择 1.4.1 创建的对象存储服务（OBS）路径中的 output 目录
- 点击提交按钮。

1.4.3 数据集智能标注

步骤 1 创建标注任务

点击数据准备 > 数据标注，点击“创建标注作业”，进入标注作业创建页面。

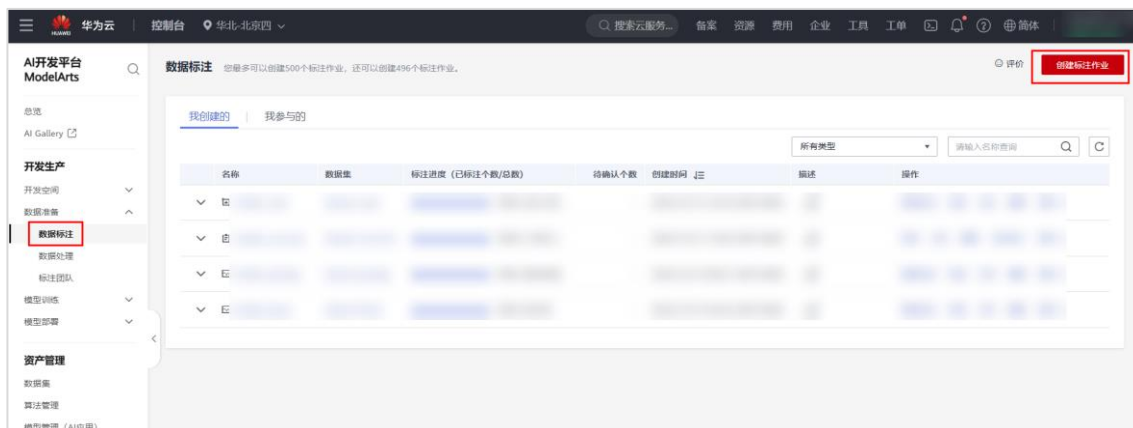


图1-10 创建标注作业

标注作业创建页面，如下图所示：

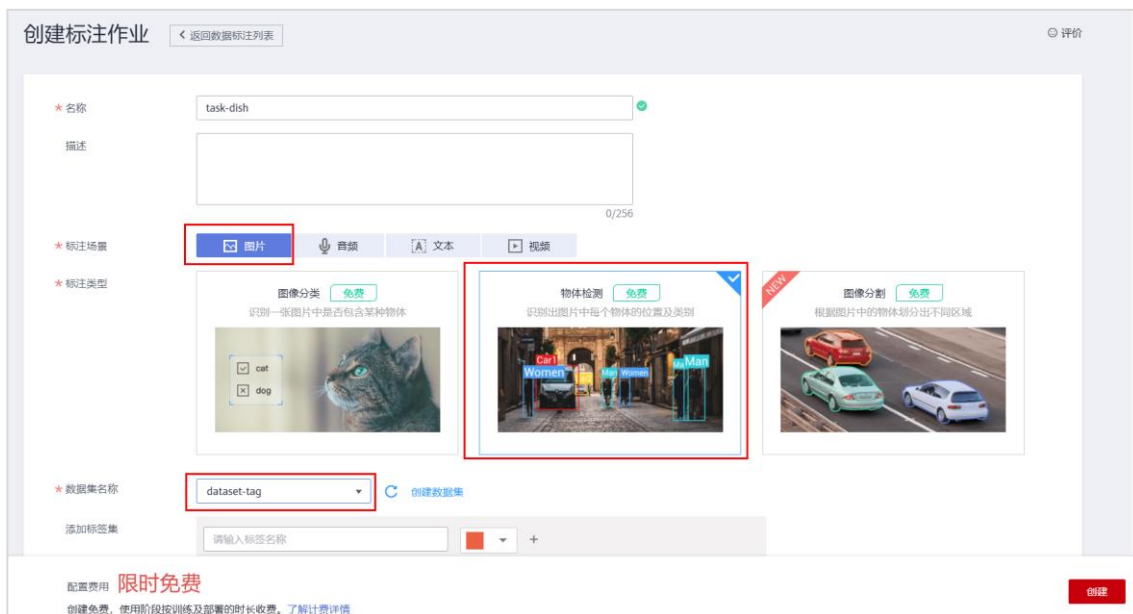


图1-11 标注作业创建页面

配置参数如下：

- 名称：自定义，便于区分
- 描述：自定义
- 标注场景：图片
- 标注类型：物体检测
- 数据集名称：选择 1.4.2 中创建的数据集

点击创建按钮。

步骤 2 数据标注

点击创建好的标注作业名称进入标注任务页面，如下图所示：

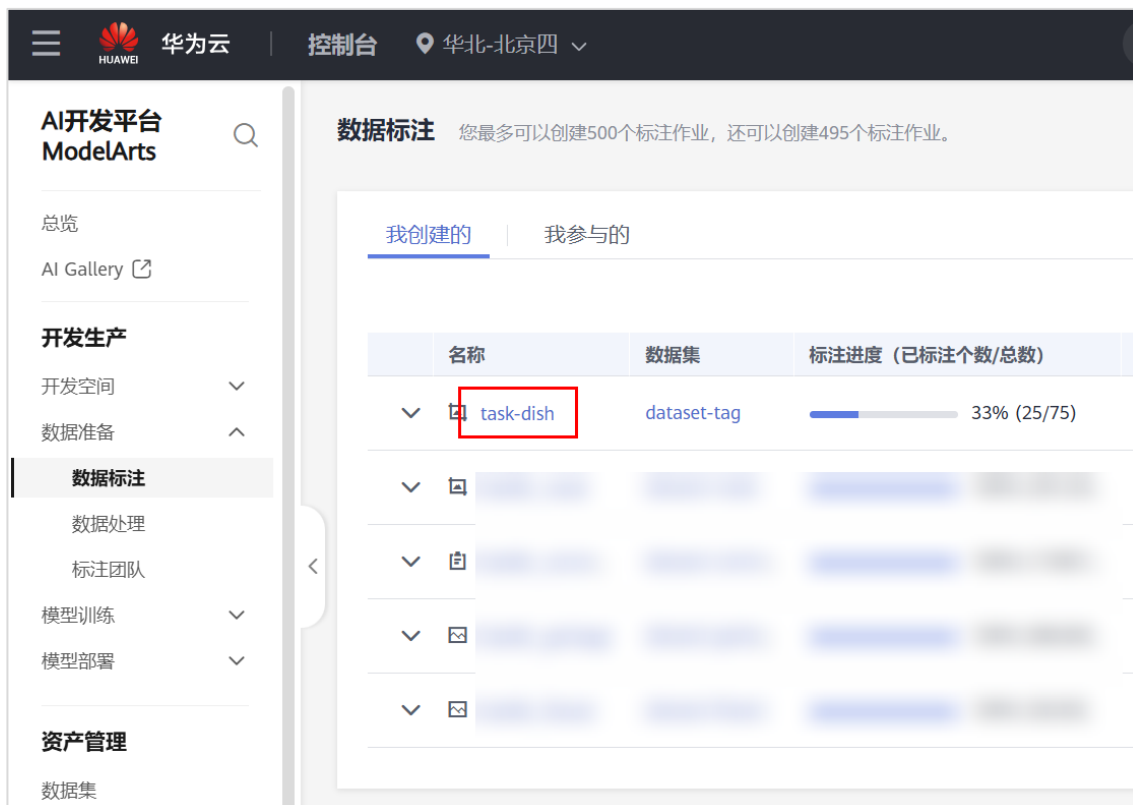


图1-12 进入标注任务页面

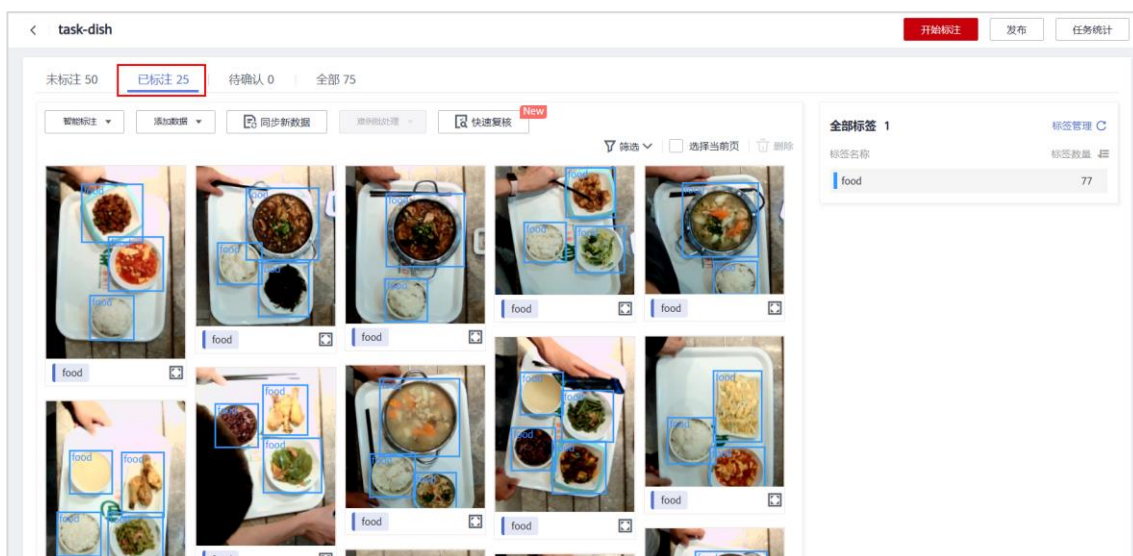


图1-13 标注任务页面

ModelArts 提供的基于主动学习的智能标注功能，可以自动为标注者挑选最具有信息量的数据，从而减少整体标注工作量。

基于主动学习的智能标注，在标注任务开始时，标注者仅需标注少量的数据作为训练集来训练模型，然后用训练好的模型对未标注数据进行推理。主动学习策略根据当前这一轮的推理结果来选择下一轮需要人工标注的数据，标注者在标注完这些数据以后将其加入训练集中，依次循环，直到模型的效果达到用户的要求。ModelArts 主动学习算法包括监督模式和半监督模式。监督算法只使用

用户已标注的数据进行训练；而半监督算法同时使用已标注数据和未标注数据，虽然可以提升模型精度但一般耗时较长。

基于当前标注阶段的标签数据及图片数据学习训练，进行智能标注，快速完成剩余图片的标注操作。

下拉“智能标注”按钮，点击“启动智能标注”。

配置参数如下：

- 智能标注类型：主动学习
- 算法类型选择：快速型
- 计算节点规格：GPU

如下图所示：

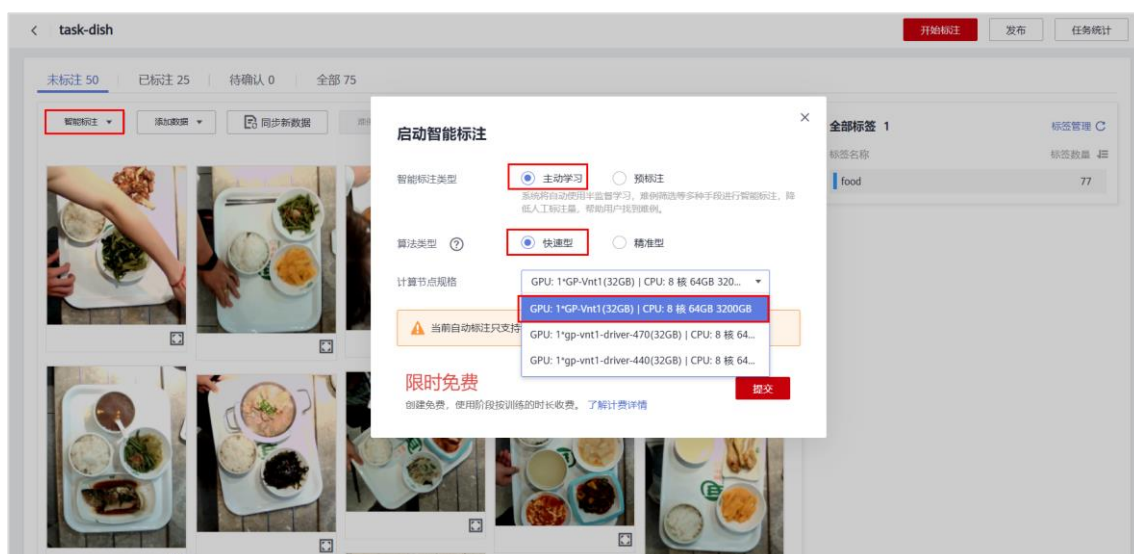


图1-14 启动智能标注

参数	说明
智能标注类型	“主动学习”。“主动学习”表示系统将自动使用半监督学习、难例筛选等多种手段进行智能标注，降低人工标注量，帮助用户找到难例。
算法类型	针对“图像分类”类型的数据集，您需要选择以下参数。 “快速型”：仅使用已标注的样本进行训练。 “精准型”：会额外使用未标注的样本做半监督训练，使得模型精度更高。
计算节点规格	即智能标注任务使用的资源规格，仅支持GPU规格，且免费公测。
计算节点个数	默认为1，表示单机模式。目前仅支持此参数值。

图1-15 参数说明

点击提交，进行智能标注。智能标注任务启动后，可以在“待确认”页签下查看智能标注任务进度。预计需等待 5 分钟（快速型智能标注），数据越多，任务运行时长越长，请耐心等待。

当任务完成后，即可在“待确认”页签下查看自动标注好的数据。



图1-16 查看自动标注数据

在智能标注任务完成后，在“待确认”页签下，单击具体图片进入标注详情页面，可以查看或修改智能标注的结果。

如果智能标注的数据无误，可单击右侧的“确认标注”，如果标注信息有误，可选中标注框进行修改，或直接删除错误标注框，然后重新标注，以纠正标注信息。针对物体检测任务，需一张一张确认。如下图所示，确认结果，确保所有图片已完成确认，然后执行下一步操作。

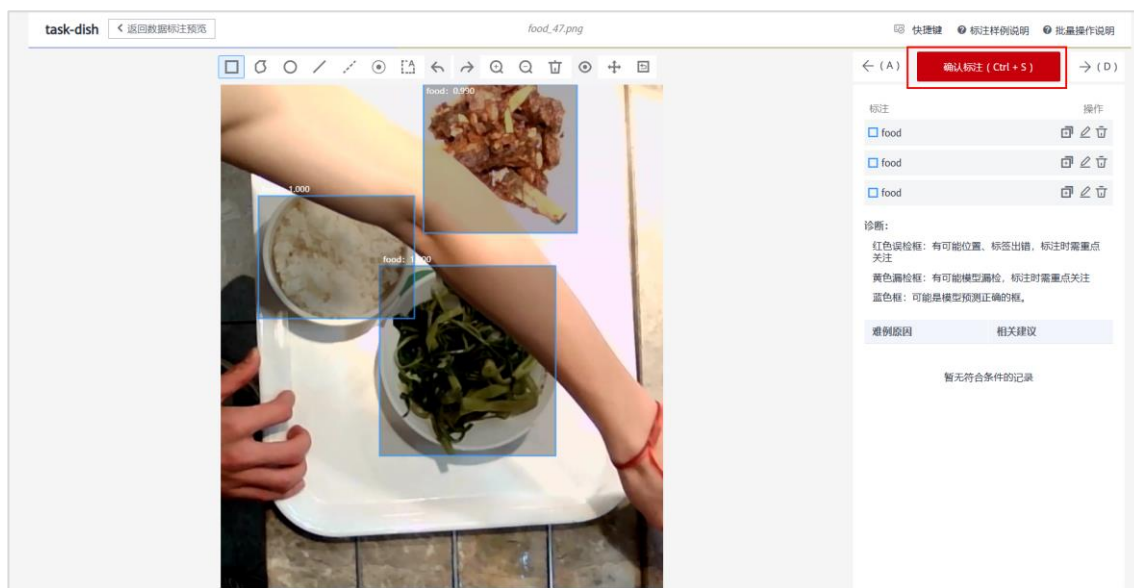


图1-17 确认标注结果

如果有一些图片没有识别出检测目标，则需自己手动标注，点击矩形按钮，标注完后选择 food 标签，点击添加，最后再点击确认标注即可。如下图所示：

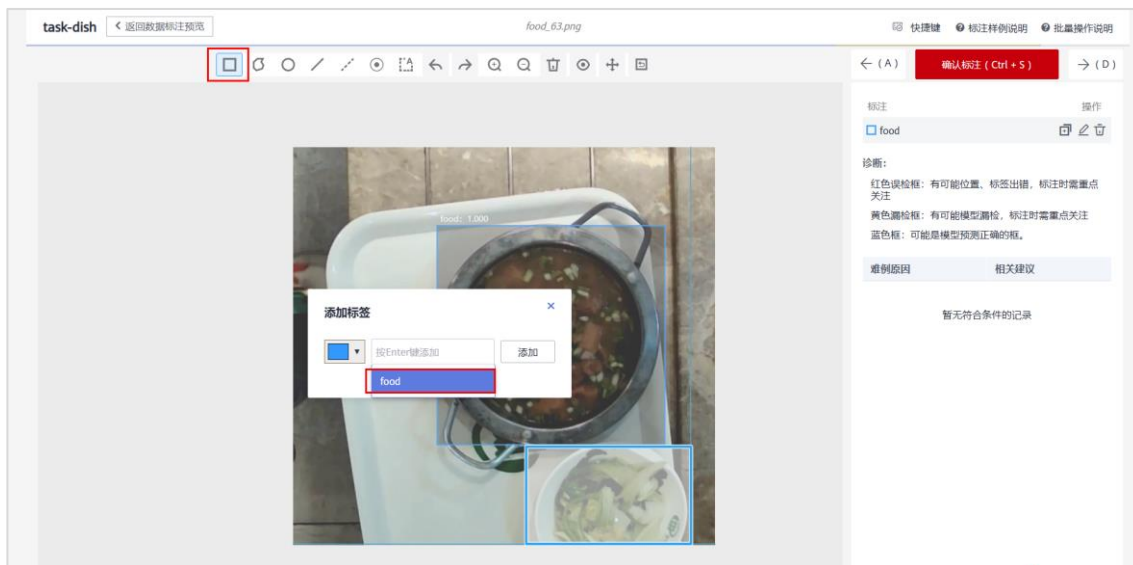


图1-18 手动添加标注

由于智能标注是基于前面个人标注行为来分析数据，所以个人标注得越准确，智能标注也就越准确，但智能标注难免会有一些误检，对于误检，点击右侧删除标识符，然后确认标注。如下图所示：

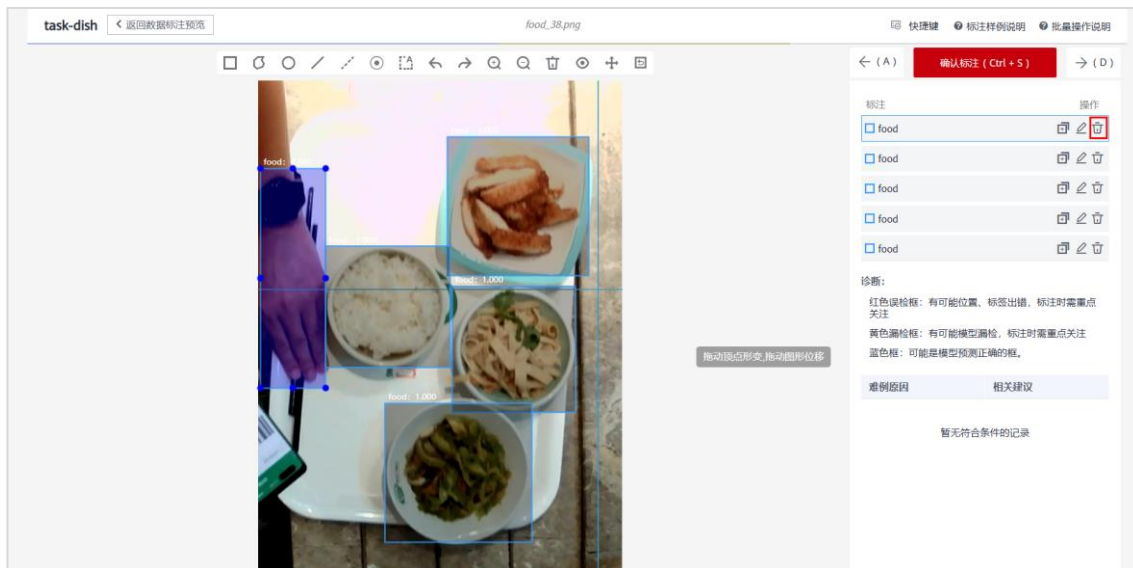


图1-19 删除错误标注

1.4.4 数据特征分析

在数据准备阶段，光有标注也是不够的，还需要对数据进行整体的统计分析，以及对单个数据进行细粒度的分析诊断，才可以更深入地了解数据，及时发现更深层次的问题并进行优化。

特征分析的主要作用在于帮助开发者快速方便地了解数据集的特点，然后制定后续的优化和处理的方案。数据集的特征分析可以融入项目开发的各个流程，如数据清洗、数据增强、模型训练、模型评估等。以目标检测为例，ModelArts 特征分析模块支持的主要特征涵盖了分辨率、图像高宽比、图像亮度、图像饱和度、清晰度、图像色彩丰富度等常规图像特征，以及目标框个数、面积标准差、堆叠度等标注相关的特征。

步骤 1 创建特征分析任务

在执行特征分析前，需先发布一个数据集版本，作为数据校验算法的输入。在智能标注中确认完数据后，单击左上角“返回数据标注预览”，然后单击“发布”，进行数据切分，“版本名称”：系统默认以 V001 开始，您也可以自行定义版本名称，发布一个新版本，如下图所示。

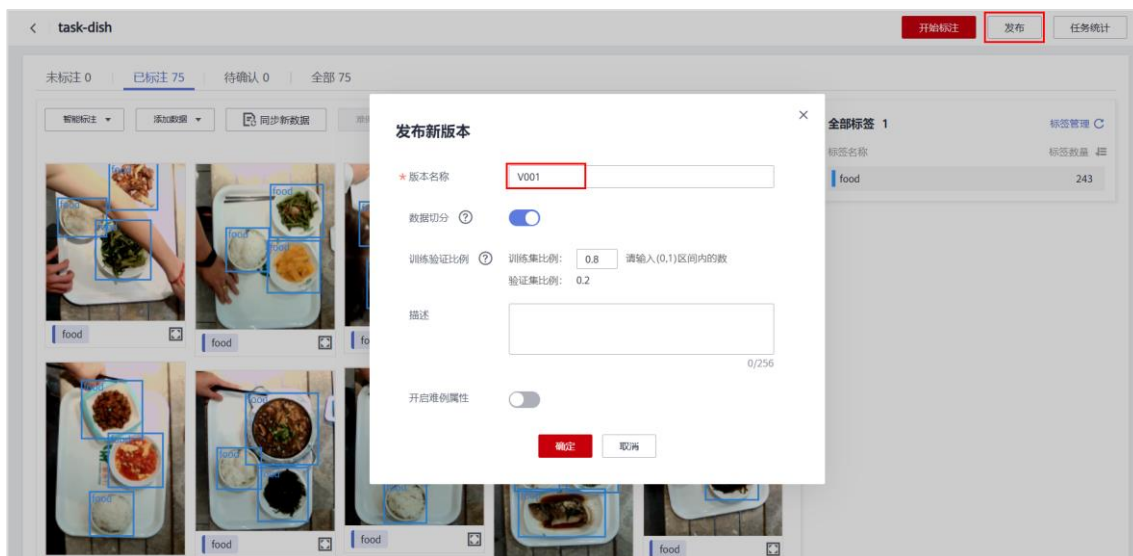


图1-20 发布数据集

版本发布完成后，进入数据集概览页。选择“数据特征”页签，单击“启动特征分析”，在弹窗中选择刚才发布的数据集版本，并单击“确定”，启动特征分析任务。



图1-21 启动特征分析

任务执行过程中，可以单击“任务历史”，查看任务进度。当任务状态变为“成功”时，表示任务执行完成。



图1-22 查看任务进度

步骤 2 查看特征分析结果

特征分析任务执行完成后，可以在“数据特征”页签下，选择“数据集版本”、“类型”和“数据特征指标”，页面将自动呈现您选择对应版本及其指标数据，您可以根据呈现的图表了解数据分布情况，帮助您更好的理解您的数据。

数据集特征分析的界面如下图所示，用户可以自行选择感兴趣的指标进行查看。除可以查看单个数据集的特征统计外，ModelArts 还支持对比功能，比如数据集不同版本之间的对比、训练集与验证集之间分布差异。通过追踪每个特征上两个数据集之间的差异，开发者可以对数据集差异情况有更好的理解并做出优化改进。例如，可以利用迁移学习来优化算法，使得模型可以自适应不同的数据分布情况。



图1-23 查看特征分析结果

- 版本选择：根据实际情况选择已执行过特征任务的版本，可以选多个进行对比，也可以只选择一个
- 类型：根据需要分析的类型选择。支持“all”、“train”、“eval”和“inference”。分别表示所有、训练、评估和推理类型
- 数据特征指标：选择您需要展示的指标。详细指标解释，可参见特征分析指标列表：<https://support.huaweicloud.com/usermanual-standard-modelarts/dataprep-prepare-modelarts-0024.html>

在特征分析结果中，例如图片亮度指标，在数据分布中，分布不均匀，缺少某一种亮度的图片，而此指标对模型训练非常关键。此时可选择增加对应亮度的图片，让数据更均衡，为后续模型构建做准备。

1.5 实验小结

本实验讲述了使用 ModelArts 实现数据集创建、智能标注和统计特征分析等功能。通过该实验，学员能更好地掌握 ModelArts 中进行智能标注、特征分析等操作的方法。

1.6 思考题

请思考：请分析说明以下特征指标：按面积占比统计框数量的分布？

答案：横坐标：目标框的面积占比，即目标框的面积占整个图片面积的比例，越大表示物体在图片中的占比越大。纵坐标：框数量（统计所有图片中的框）。主要判断模型中使用的 anchor 的分布，如果目标框普遍较大，anchor 就可以选择较大。

2 深度学习预置算法

2.1 实验简介

ModelArts 提供了包括图像分类，目标检测在内的一系列预置模型，包括经典的 resnet 和轻量级模型 mobilenet 等等。预置算法能极大地缩短在新的数据集上训练的耗时，并且精度也会比较高，是深度学习常用的训练方法。

2.2 实验目的

指导 ModelArts 用户简便地使用预置算法，让使用者只需要准备符合要求格式的数据集就能进行训练。

2.3 实验步骤

2.3.1 垃圾分类应用

步骤 1 准备训练数据

在 2.3.1 垃圾分类应用 > 步骤 1 中已经创建好了 garbage 数据集，并发布了数据集版本。本实验使用之前创建好的 garbage 数据集即可。

步骤 2 订阅算法

进入“ModelArts”管理控制台界面。单击左侧导航栏的“AI Gallery”，进入 AI 市场界面。



图2-1 进入 AI Gallery

在 AI Gallery 进入模型列表，选择模型标签为“图像分类”，点击“图像分类 ResNet50-EI-Backbone”，进入算法详情页。

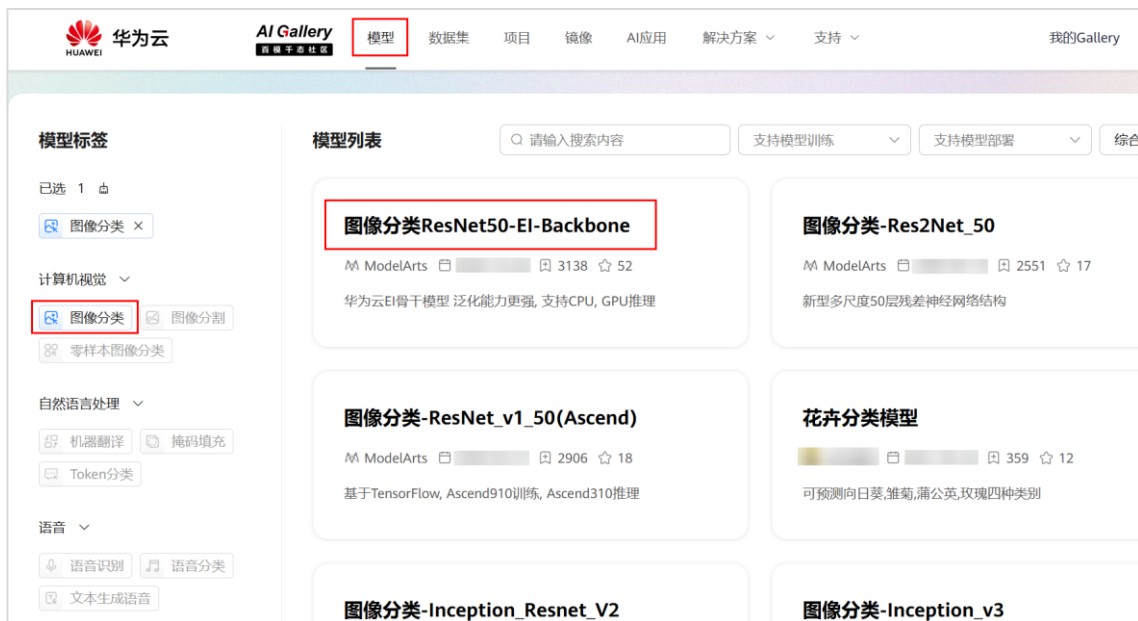


图2-2 选择算法

单击右侧的“训练 > ModelArts”后，选择 ModelArts 的云服务区域（华北-北京四），单击“确认”，跳转至 ModelArts 的“算法管理 > 订阅算法”中。

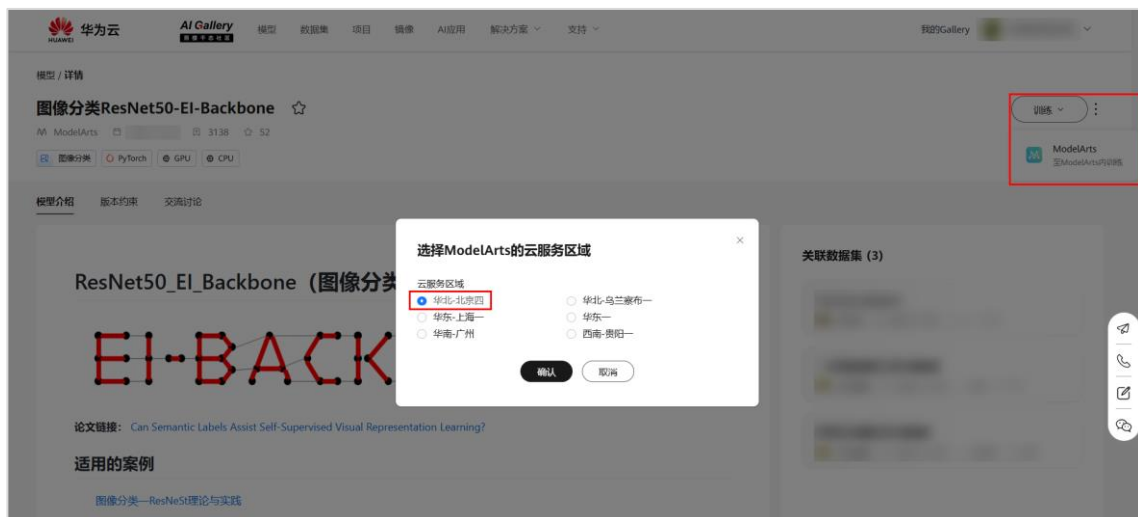


图2-3 订阅算法

步骤 3 使用订阅算法创建训练作业

算法订阅成功后，算法将呈现在“算法管理 > 订阅算法”中，您可以使用订阅的算法创建训练作业，获得模型。

进入“算法管理 > 订阅算法”页面，选择订阅的“图像分类 ResNet50-EI-Backbone”算法，单击操作列的“创建训练作业”。

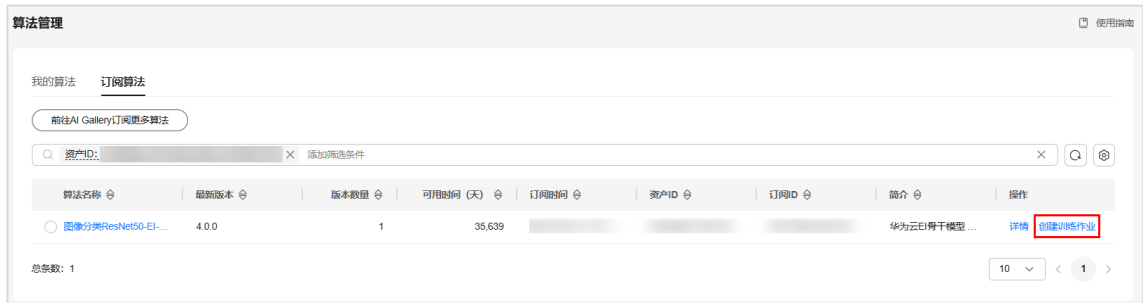
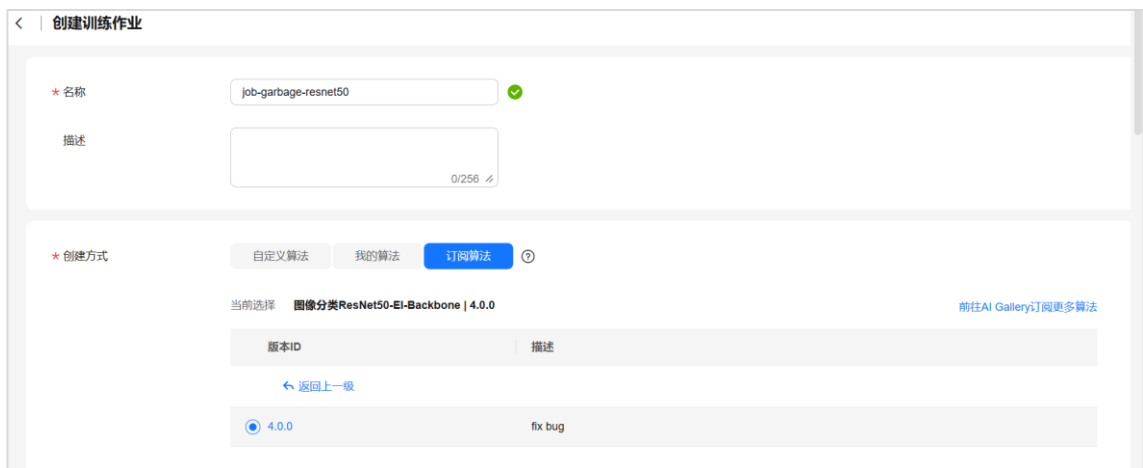


图2-4 创建训练作业

在创建训练作业页面，参考如下说明填写关键参数：

- 创建方式：系统默认选择订阅的算法，请勿随意修改
- 输入：选择之前创建好的 garbage 数据集
- 输出：选择一个 OBS 空目录存储训练输出的模型，例如：“modelarts-exp/pretrained-model/garbage-resnet50”
- 超参：建议采用默认值
- 资源池：公共资源池
- 资源类型：可以选择限时免费的 GPU 规格资源，如果希望训练效率更高，可以选择收费的 GPU 资源
- 实例数：建议采用默认值 1
- 作业日志路径：与训练输出路径保持一致即可，例如：“modelarts-exp/pretrained-model/garbage-resnet50”



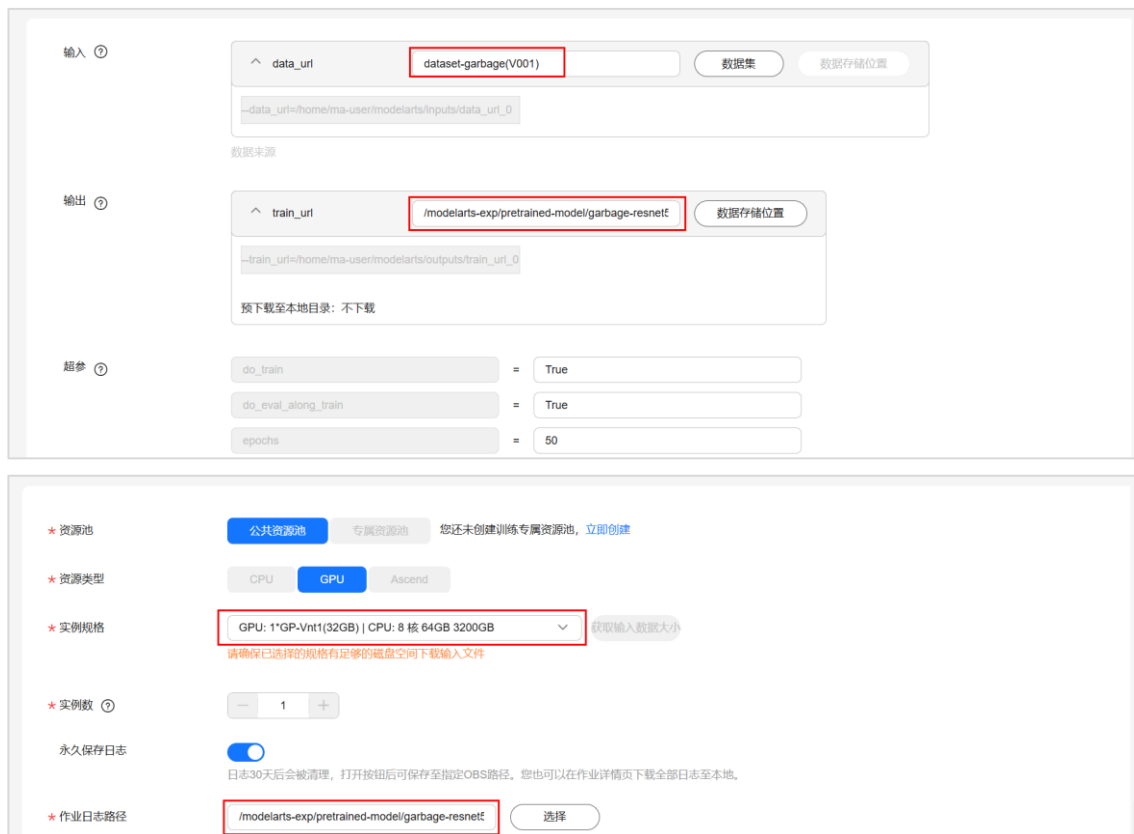
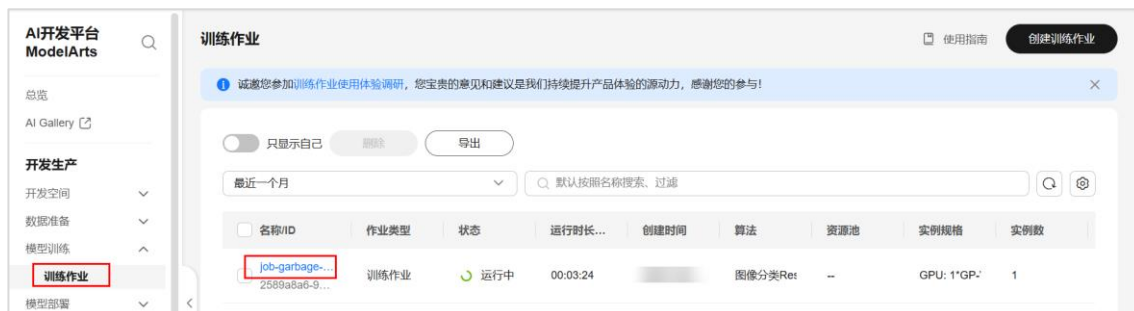


图2-5 创建训练作业页面

参数填写完成后，单击“提交”，根据界面提示确认规格，单击“确定”，完成训练作业创建。进入“模型训练 > 训练作业”页面，等待训练作业完成。



名称/ID	作业类型	状态	运行时长...	创建时间	算法	资源池	实例规格	实例数
job-garbage-... 2589a8a6-9...	训练作业	运行中	00:03:24		图像分类Ret		GPU: 1*GP...	1

图2-6 训练作业页面

训练作业运行需要几分钟时间，请耐心等待。根据经验，选择样例数据集，使用 GPU 资源运行，预计 10 分钟左右可完成。

当训练作业的状态变更为“已完成”时，表示已运行结束。

您可以单击训练作业名称，进入详情页面，了解训练作业的“配置信息”、“日志”、“资源占用情况”和“评估结果”等信息。您也可以在配置的“输出”对应的 OBS 目录下获得训练生成的模型。

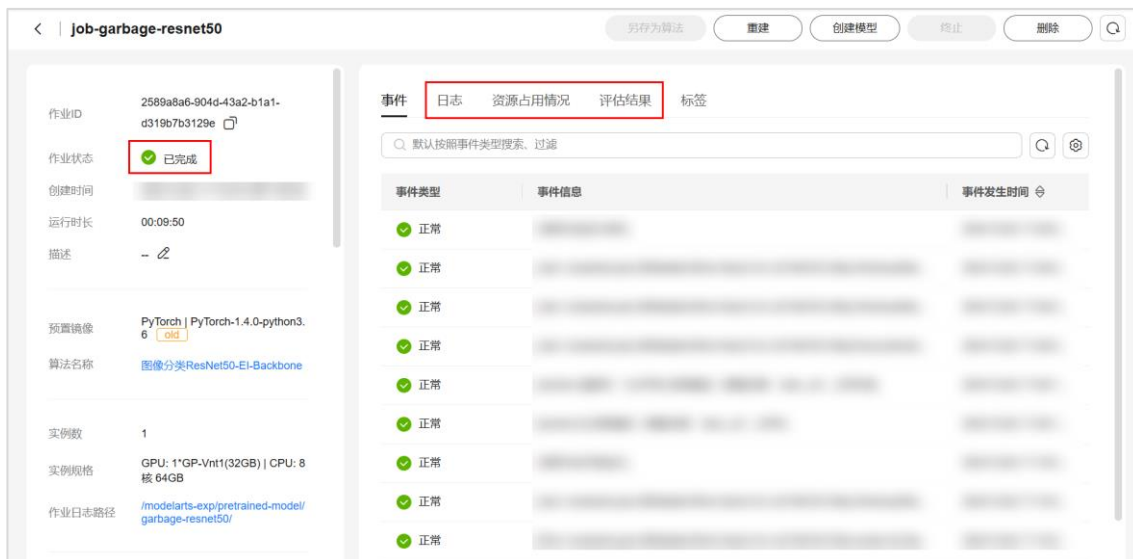


图2-7 训练作业详情页面

步骤 4 创建模型（AI 应用）

在训练作业详情页的右上角单击“创建模型”，进入创建模型页面。

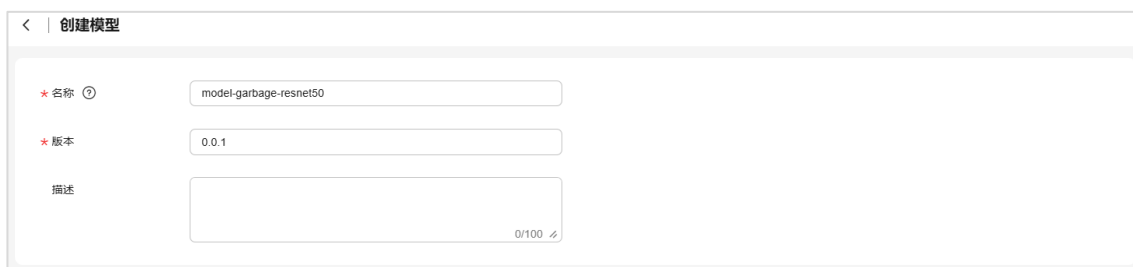
也可以在 ModelArts 管理控制台，选择“资产管理 > 模型管理（AI 应用）”，在“自定义模型”页面，单击“创建模型”，进入创建模型页面。



图2-8 创建模型

在创建模型页面，系统会自动根据上一步训练作业填写参数，参考如下说明确认关键参数。

- 元模型来源：系统自动选择“从训练中选择”
- 选择训练作业：系统自动选择上一步创建的训练作业
- AI 引擎：系统自动写入该模型的 AI 引擎，无需修改
- 推理代码：系统自动放置推理代码到 OBS 输出路径，无需修改
- 部署类型：默认选择“在线服务”



★ 元模型来源 ②

从训练中选择 从对象存储服务（OBS）中选择 从容器镜像中选择

1. 导入ModelArts训练作业中训练完成的模型。请在下方选择需要导入的训练作业。模型规范详情参见 [从训练中选择元模型](#)。
2. 使用容器化部署，导入的元模型有大小限制，详情请参见 [导入模型对于镜像大小限制](#)。
3. 请确认选择正确的训练作业，避免因算法类型错误而导致预测结果不正确。

★ 选择训练作业 job-garbage-resnet50

☐ 动态加载 ②

★ AI引擎 PyTorch pytorch1.4-python3.7

推理代码 /modelarts-exp/pretrained-model/garbage-resnet50/model/customize_service.py

运行时依赖 ②

安装方式	名称	版本	约束
pip	torchvision	0.5.0	当前版本
pip	Pillow	5.2.0	当前版本

模型说明 ② [添加模型说明](#)

★ 部署类型 ② ☒ 在线服务 ☐ 批量服务 ☐ 边缘服务

图2-9 创建模型页面

参数填写完成后，单击“立即创建”。自动跳转至自定义模型列表页面，等待创建结果，预计 2 分钟左右。

当模型的状态变为“正常”时，表示创建成功。

步骤 5 部署为在线服务

AI 应用创建成功后，可将其部署为在线服务，在部署时可使用 CPU 资源。

单击模型名称，进入详情页面，单击“部署 > 在线服务”。

< 返回模型 0.0.1

名称 model-garbage-resnet50 状态 正常

版本 0.0.1 ID 52631565-5054-4427-8e2f-32204e63588f

描述 -- 部署类型 在线服务

元模型来源 从训练中选择 训练作业名称 job-garbage-resnet50

AI引擎 PyTorch 运行环境 pytorch1.4-python3.7

容器调用接口 ② https://[host]:8443 推理代码 /modelarts-exp/pretrained-model/garbage-resnet...

动态加载 ② 否 大小 90.00 MB

模型说明 --

参数配置 运行时依赖 事件 使用约束 关联服务

[查看模型apis定义](#)

POST

部署 在线服务 边缘服务

模型精度 召回率 精确率 准确率 F1值

图2-10 部署在线服务

在部署页面，参考如下说明填写关键参数：

- 资源池：选择“公共资源池”
- 选择模型及版本：模型来源及版本会自动选择前面创建的模型

- 实例规格：在下拉框中选择限时免费的 CPU 资源，如果限时免费资源售罄，建议选择收费 CPU 资源进行部署
- 实例数：默认设置为 1
- 其他参数可使用默认值



图2-11 在线服务详情

参数设置完成后，单击“下一步”，确认规格参数，单击“提交”，完成在线服务的部署。

步骤 6 服务测试

您可以进入“模型部署 > 在线服务”页面，等待服务部署完成，当服务状态变为“运行中”时，表示服务部署成功。预计时长 2 分钟左右。

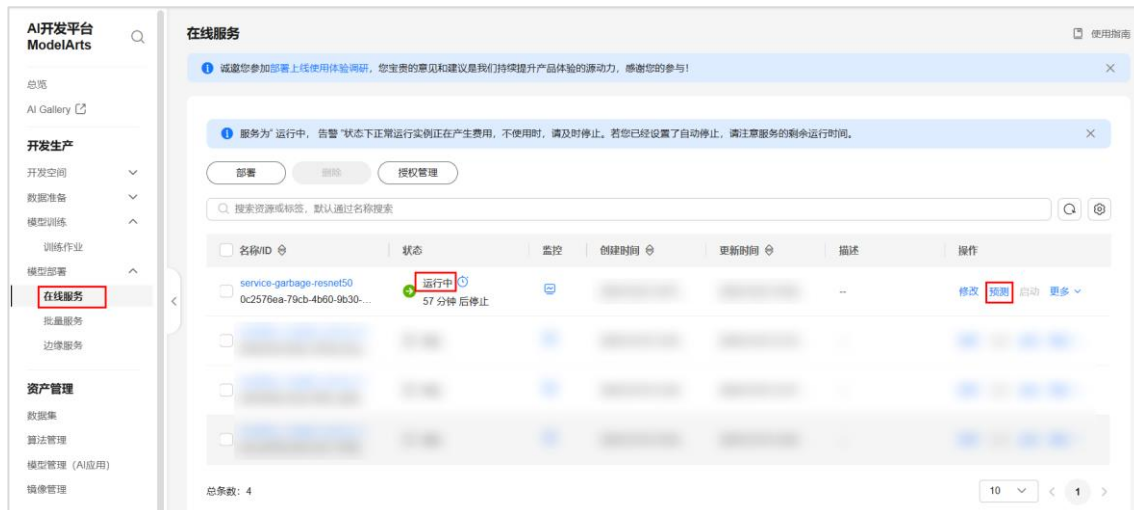


图2-12 服务部署成功

在线服务部署完成后，您可以单击操作列的“预测”，进入服务详情页的“预测”页面。

在“预测”页签，单击“上传”，上传一个测试图片，单击“预测”进行预测。



图2-13 服务测试

步骤 7 清除资源

为避免产生不必要的费用，通过此示例学习订阅算法的使用后，建议您清除相关资源，避免造成资源浪费。

- 停止在线服务：在“在线服务”页面，单击对应服务操作列的“停止”
- 删除训练作业：在“训练作业”页面，选中相应训练作业后单击“删除”
- 删除数据：前往 OBS，删除数据，然后删除文件夹及 OBS 桶

2.3.2 口罩检测应用

步骤 1 准备训练数据

在 2.3.2 垃圾分类应用 > 步骤 1 中已经创建好了 mask 数据集，并发布了数据集版本。本实验使用之前创建好的 mask 数据集即可。

步骤 2 订阅算法

进入“ModelArts”管理控制台界面。单击左侧导航栏的“AI Gallery”，进入 AI 市场界面。



图2-14 进入 AI Gallery

在 AI Gallery 进入模型列表，选择模型标签为“目标检测”，点击“物体检测 YOLOv3_ResNet18”，进入算法详情页。

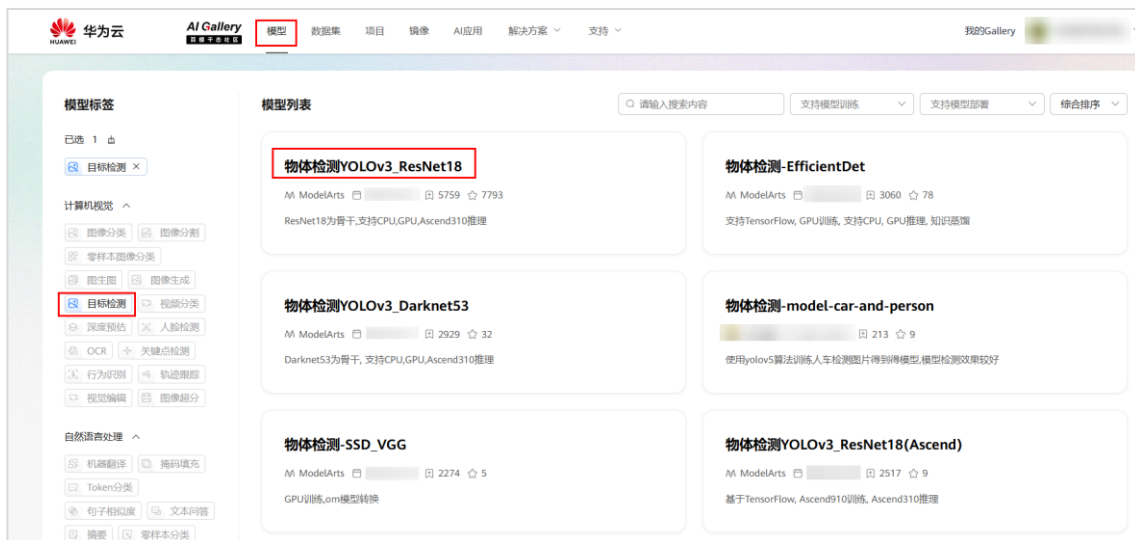


图2-15 选择算法

单击右侧的“训练 > ModelArts”后，选择 ModelArts 的云服务区域（华北-北京四），单击“确认”，跳转至 ModelArts 的“算法管理 > 订阅算法”中。

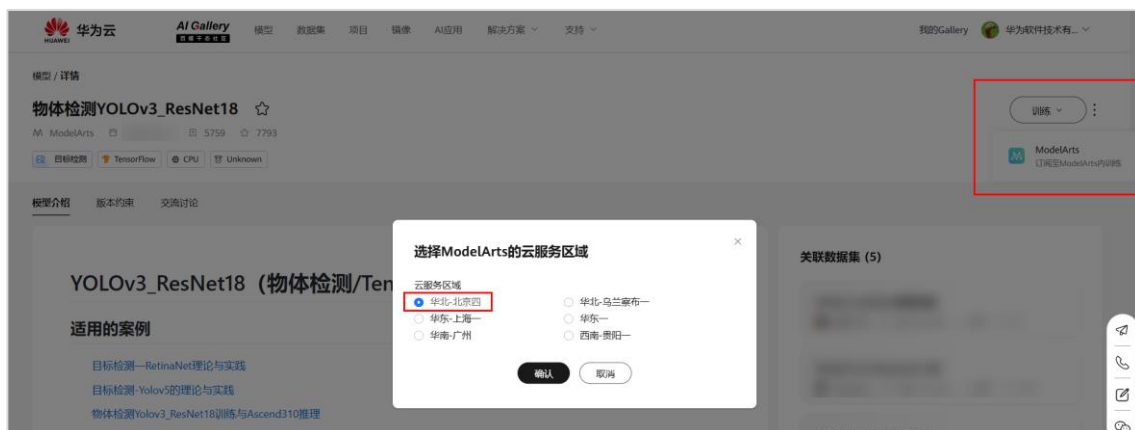


图2-16 订阅算法

步骤 3 使用订阅算法创建训练作业

算法订阅成功后，算法将呈现在“算法管理 > 订阅算法”中，您可以使用订阅的算法创建训练作业，获得模型。

进入“算法管理 > 订阅算法”页面，选择订阅的“物体检测 YOLOv3_ResNet18”算法，单击操作列的“创建训练作业”。

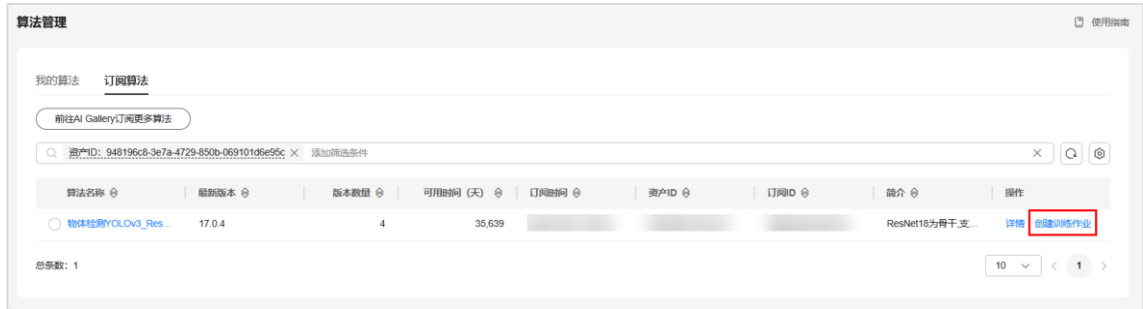


图2-17 创建训练作业

在创建训练作业页面，参考如下说明填写关键参数：

- 创建方式：系统默认选择订阅的算法，请勿随意修改
- 输入：选择之前创建好的 mask 数据集
- 输出：选择一个 OBS 空目录存储训练输出的模型，例如：“modelarts-exp/pretrained-model/mask-yolov3”
- 超参：建议采用默认值
- 资源池：公共资源池
- 资源类型：可以选择限时免费的 GPU 规格资源，如果希望训练效率更高，可以选择收费的 GPU 资源
- 实例数：建议采用默认值 1
- 作业日志路径：与训练输出路径保持一致即可，例如：“modelarts-exp/pretrained-model/mask-yolov3”



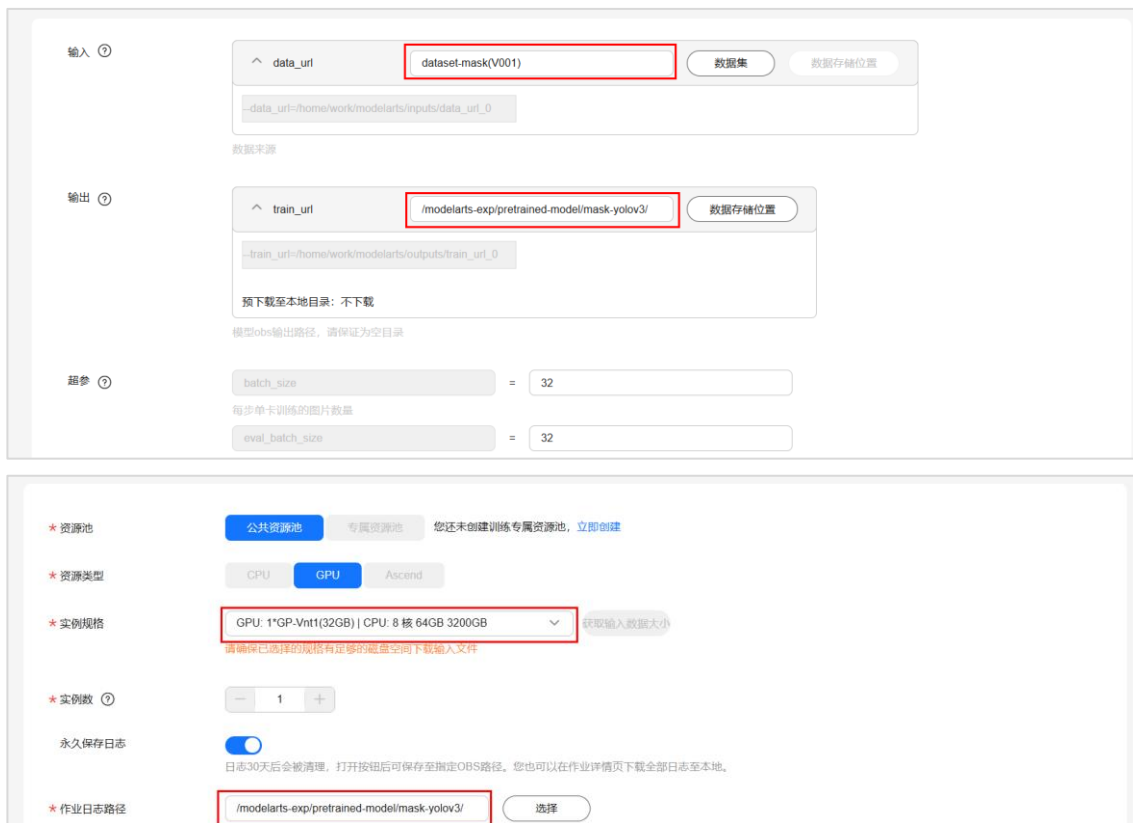


图2-18 创建训练作业页面

参数填写完成后，单击“提交”，根据界面提示确认规格，单击“确定”，完成训练作业创建。进入“模型训练 > 训练作业”页面，等待训练作业完成。



名称/ID	作业类型	状态	运行时长...	创建时间	算法	资源池	实例规格	实例数	作业优先级	描述
job-mask-yolov3	训练作业	运行中	00:01:36		物体检测YO		GPU: 1*GP-Vnt(32GB) CPU: 8 核 64GB 3200GB	1		

图2-19 训练作业页面

训练作业运行需要几分钟时间，请耐心等待。根据经验，选择样例数据集，使用 GPU 资源运行，预计 10 分钟左右可完成。

当训练作业的状态变更为“已完成”时，表示已运行结束。

您可以单击训练作业名称，进入详情页面，了解训练作业的“配置信息”、“日志”、“资源占用情况”和“评估结果”等信息。您也可以在配置的“输出”对应的 OBS 目录下获得训练生成的模型。

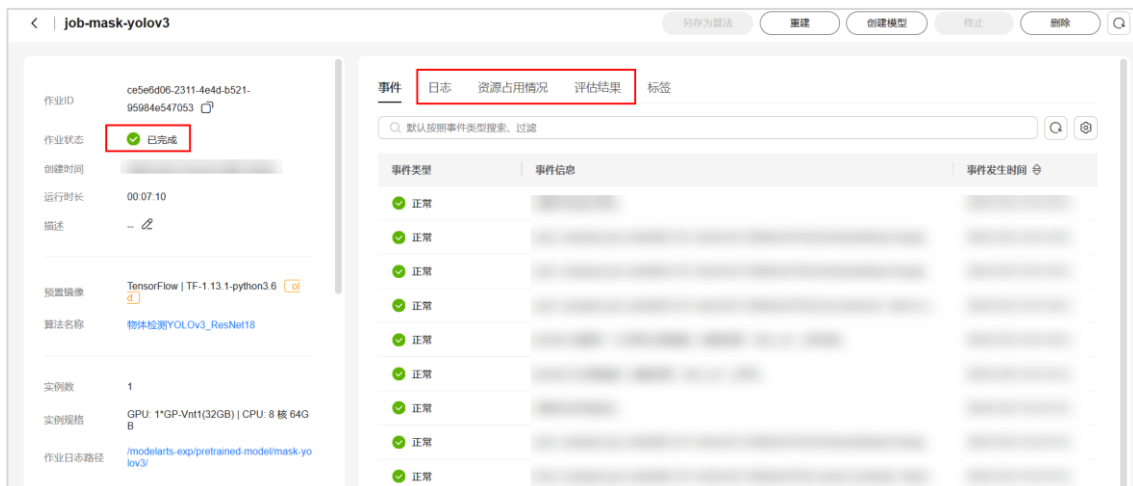


图2-20 训练作业详情页面

步骤 4 创建模型（AI 应用）

在训练作业详情页的右上角单击“创建模型”，进入创建模型页面。

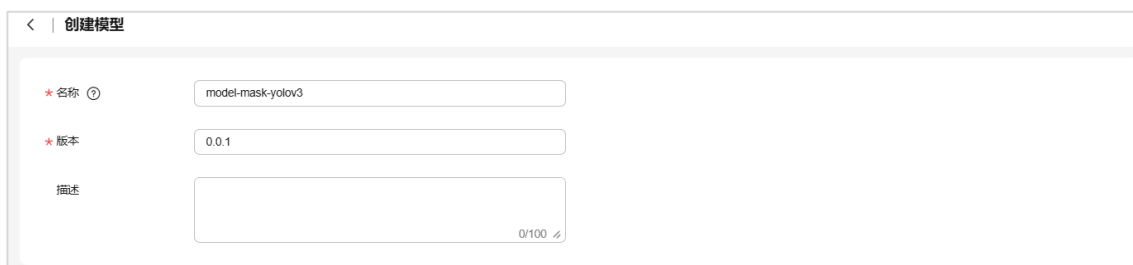
也可以在 ModelArts 管理控制台，选择“资产管理 > 模型管理（AI 应用）”，在“自定义模型”页面，单击“创建模型”，进入创建模型页面。



图2-21 创建模型

在创建模型页面，系统会自动根据上一步训练作业填写参数，参考如下说明确认关键参数。

- 元模型来源：系统自动选择“从训练中选择”
- 选择训练作业：系统自动选择上一步创建的训练作业
- AI 引擎：系统自动写入该模型的 AI 引擎，无需修改
- 推理代码：系统自动放置推理代码到 OBS 输出路径，无需修改
- 部署类型：默认选择“在线服务”



*** 元模型来源** ①

从训练中选择 | 从对象存储服务 (OBS) 中选择 | 从容器镜像中选择

1. 导入ModelArts训练作业中训练完成的模型。请在下方选择需要导入的训练作业。模型规范详情参见 [从训练中选择元模型](#)。
2. 使用容器化部署，导入的元模型有大小限制，详情请参见 [导入模型对于镜像大小限制](#)。
3. 请确认选择正确的训练作业，避免因算法类型错误而导致预测结果不正确。

*** 选择训练作业** job-mask-yolov3

☐ 动态加载 ①

*** AI引擎** TensorFlow | tf1.13-python3.7-cpu

推理代码 /modelarts-exp/pretrained-model/mask-yolov3/model/customize_service.py

运行时依赖 ①

安装方式	名称	版本	约束
pip	numpy	1.17.0	当前版本
pip	h5py	2.8.0	当前版本
pip	Pillow	5.2.0	当前版本
pip	scipy	1.2.1	当前版本
pip	resampy	0.2.1	当前版本
pip	scikit-learn	0.22.2	当前版本

模型说明 ① [添加模型说明](#)

*** 部署类型** ① ☒ 在线服务 ☐ 批量服务 ☐ 边缘服务

图2-22 创建模型页面

参数填写完成后，单击“立即创建”。自动跳转至自定义模型列表页面，等待创建结果，预计2分钟左右。

当模型的状态变为“正常”时，表示创建成功。

步骤 5 部署为在线服务

AI应用创建成功后，可将其部署为在线服务，在部署时可使用CPU资源。

单击模型名称，进入详情页面，单击“部署 > 在线服务”。

返回模型 0.0.1

部署 **在线服务** **删除**

名称	model-mask-yolov3	状态	正常
版本	0.0.1	ID	824c9112-abd4-4479-9352-7a41984d9f78
描述	--	部署类型	在线服务
元模型来源	从训练中选择	训练作业名称	job-mask-yolov3
AI引擎	TensorFlow	运行环境	tf1.13-python3.7-cpu
容器调用接口 ①	https://[host]:8443	推理代码	/modelarts-exp/pretrained-model/mask-yolov3/m...
动态加载 ①	否	大小	63.18 MB
模型说明	--		

模型精度

召回率 0.9331499999999999

精确率 0.9331499999999999

准确率 0.9331499999999999

F1值 0.8182225799656967

参数配置 运行时依赖 事件 使用约束 关联服务

[查看模型API定义](#)

POST /

图2-23 部署在线服务

在部署页面，参考如下说明填写关键参数：

- 资源池：选择“公共资源池”
- 选择模型及版本：模型来源及版本会自动选择前面创建的模型
- 实例规格：在下拉框中选择限时免费的 CPU 资源，如果限时免费资源售罄，建议选择收费 CPU 资源进行部署
- 实例数：默认设置为 1
- 其他参数可使用默认值

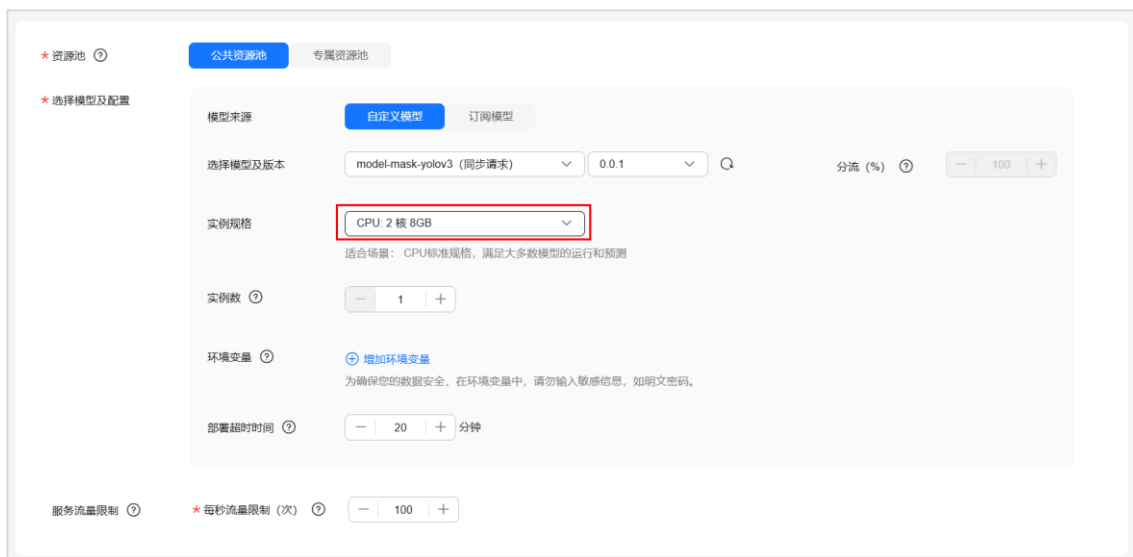


图2-24 在线服务详情

参数设置完成后，单击“下一步”，确认规格参数，单击“提交”，完成在线服务的部署。

步骤 6 服务测试

您可以进入“模型部署 > 在线服务”页面，等待服务部署完成，当服务状态变为“运行中”时，表示服务部署成功。预计时长 2 分钟左右。

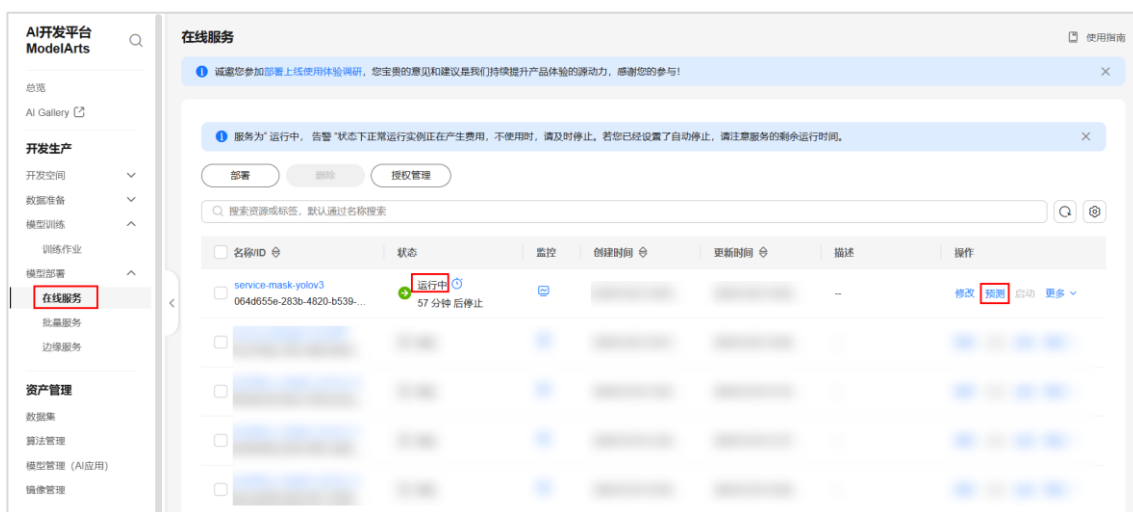


图2-25 服务部署成功

在线服务部署完成后，您可以单击操作列的“预测”，进入服务详情页的“预测”页面。
在“预测”页签，单击“上传”，上传一个测试图片，单击“预测”进行预测。



图2-26 服务测试

步骤 7 清除资源

为避免产生不必要的费用，通过此示例学习订阅算法的使用后，建议您清除相关资源，避免造成资源浪费。

- 停止在线服务：在“在线服务”页面，单击对应服务操作列的“停止”
- 删除训练作业：在“训练作业”页面，选中相应训练作业后单击“删除”
- 删除数据：前往 OBS，删除数据，然后删除文件夹及 OBS 桶

2.4 实验小结

本实验讲述了使用 ModelArts 预置模型实现图像分类和物体检测等功能。通过该实验，学员能更好地掌握快速创建图像分类和物体检测模型的方法。

2.5 思考题

请思考：预置算法和自动学习的区别？

答案：预置算法相比于自动学习自动设计模型和自动调参，可以手动选择模型和配置超参，具备更强的性能和灵活性。

华为认证 AI 系列教程

HCIP-AI-EI Developer

华为企业AI开发高级工程师认证

Vision Transformer图像分类

实验指导手册

版本：2.5



华为技术有限公司

版权所有 © 华为技术有限公司 2026。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<https://e.huawei.com>

华为认证体系介绍

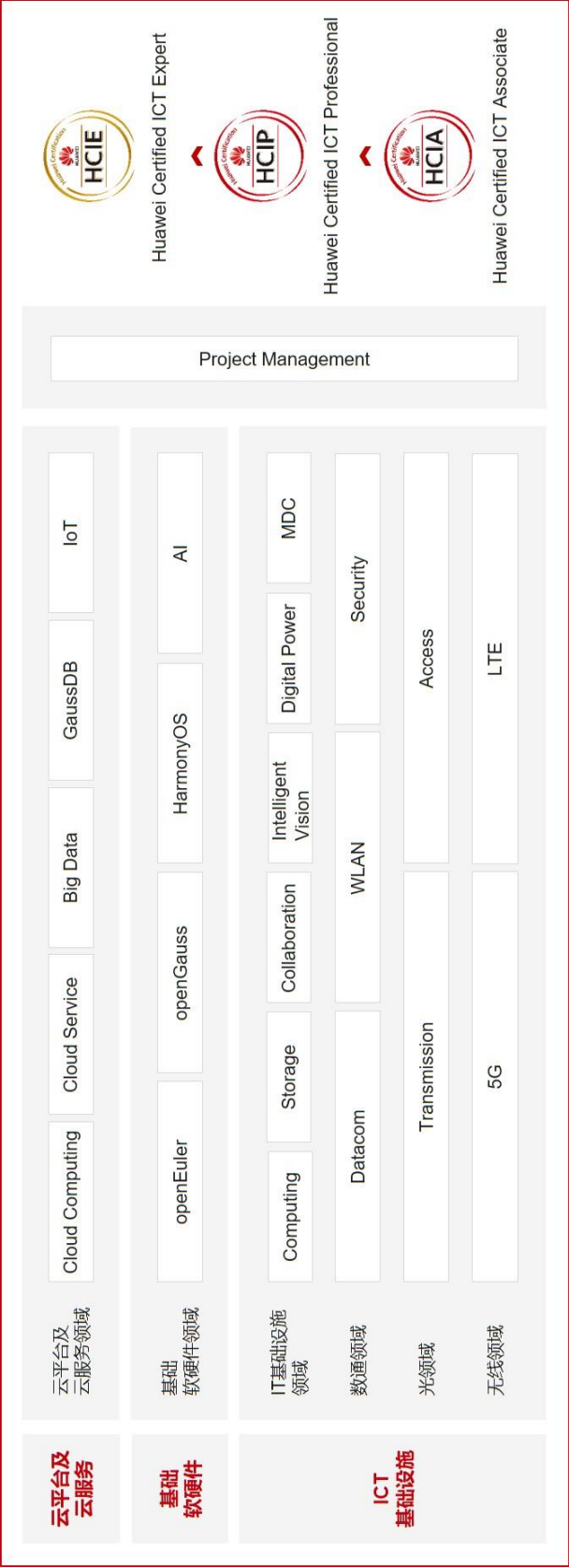
华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的覆盖ICT（Information and Communications Technology，信息通信技术）全技术领域的认证体系，包含ICT基础设施认证、基础软硬件认证、云平台及云服务认证三类认证。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。

华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIP-AI-EI Developer V2.5定位于培养与认证具有图像处理、语音处理、自然语言处理能力，能够应用通用开源框架和华为云ModelArts平台等进行开发和创新的高级工程师。

通过HCIP-AI-EI Developer V2.5认证，您将掌握神经网络基础、图像处理理论、语音处理理论、自然语言处理理论、大模型深度学习算法等基础知识，具备运用通用开源框架和华为云ModelArts平台等进行图像处理、语音处理和自然语言处理等人工智能产品设计开发的能力，能够胜任AI开发高级工程师、图像处理开发工程师、语音处理开发工程师、自然语言处理开发工程师等岗位。



简介

本书为 HCIP-AI 认证培训教程，适用于准备参加 HCIP-AI 考试的学员或者希望了解 AI 基础知识的读者。掌握本实验手册内容，您将能够实现基于 Vision Transformer 的图像应用，并深入了解 ViT 网络在计算机视觉领域的应用。

内容描述

本实验指导书将基于 PyTorch 框架实现 Vision Transformer 的图像应用。

读者知识背景

本课程为华为认证基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

- 具有基本的 Python 语言编辑能力
- 有一定自然语言处理的理论基础
- 对 PyTorch 框架一定的了解

实验环境说明

实验使用语言为 Python，框架为 PyTorch。

- Python3.9 及以上
 - torch>=2.1 (GPU、cuda11.8/12.4)
-

前 言	4
简介	4
内容描述	4
读者知识背景	4
实验环境说明	4
1 Vision Transformer 图像分类	6
1.1 实验介绍	6
1.1.1 关于本实验	6
1.1.2 实验目的	6
1.1.3 环境准备	6
1.1.4 实验总体设计	6
1.2 实验流程	7
1.2.1 实验资源获取	7
1.2.2 导入实验环境	7
1.2.3 数据处理与加载	9
1.2.4 模型构建	11
1.2.5 模型训练和评估	14
1.2.6 模型验证	17
1.2.7 模型推理	18
1.3 实验小节	20
1.4 思考题	20

1

Vision Transformer 图像分类

1.1 实验介绍

1.1.1 关于本实验

ViT (Vision Transformer) 是 2020 年 Google 团队提出的将 Transformer 应用在图像分类的模型，Transformer 在 NLP 中大获成功，Vision Transformer 则将 Transformer 模型架构扩展到计算机视觉的领域中，并且它可以很好的取代卷积操作，在不依赖卷积的情况下，依然可以在图像分类任务上达到很好的效果。

卷积操作只能考虑到局部的特征信息，而 Transformer 中的注意力机制可以综合考量全局的特征信息。Vision Transformer 尽力做到在不改变 Transformer 中 Encoder 架构的前提下，直接将其从 NLP 领域迁移到计算机视觉领域中，目的是让原始的 Transformer 模型开箱即用。

1.1.2 实验目的

本实验的主要目的是通过基于 Vision Transformer 的图像分类实验，学员能深入地了解 ViT 网络在计算机视觉领域的应用。通过案例代码的学习，重点理解 ViT 模型的 Patch Embedding 层、Transformer Encoder 模块以及 MLP 分类头。

1.1.3 环境准备

本实验在华为云 ModelArts 上进行，使用 Notebook 开发环境，基于 PyTorch 2.5 CPU/GPU, EVS 为 20GB 环境进行。具体的购买和打开操作请参考《华为云 ModelArts 开发环境搭建指南》，此处不重复赘述。

1.1.4 实验总体设计

该实验主要步骤包括：

- 数据集获取；
 - 预训练模型获取；
 - 导入实验环境；
 - 读取数据集；
 - 数据处理；
 - 模型构建；
-

- 模型训练；
- 模型评估；
- 模型预测。

1.2 实验流程

1.2.1 实验资源获取

步骤 1 预训练权重下载

在 ModelArts-Terminal 中下载预训练权重文件。

Terminal 操作指令：

```
wget https://huggingface.co/mbehbooei/vit-base-patch16-224-in21k-finetuned-moderation/resolve/main/config.json -P ./vit-base-patch16-224-ft/

#下载 pytorch_model.bin:
wget https://huggingface.co/mbehbooei/vit-base-patch16-224-in21k-finetuned-moderation/resolve/main/pytorch_model.bin -P ./vit-base-patch16-224-ft/

wget https://huggingface.co/WinKawaks/vit-tiny-patch16-224/resolve/main/config.json -P ./vit-tiny-patch16-224/
wget https://huggingface.co/WinKawaks/vit-tiny-patch16-224/resolve/main/pytorch_model.bin -P ./vit-tiny-patch16-224/
```

下载完成后，在 vit-base-patch16-224-in21k 文件夹中可以看到预训练模型文件：
pytorch_model.bin & config.json。

1.2.2 导入实验环境

步骤 1 导入相应的模块

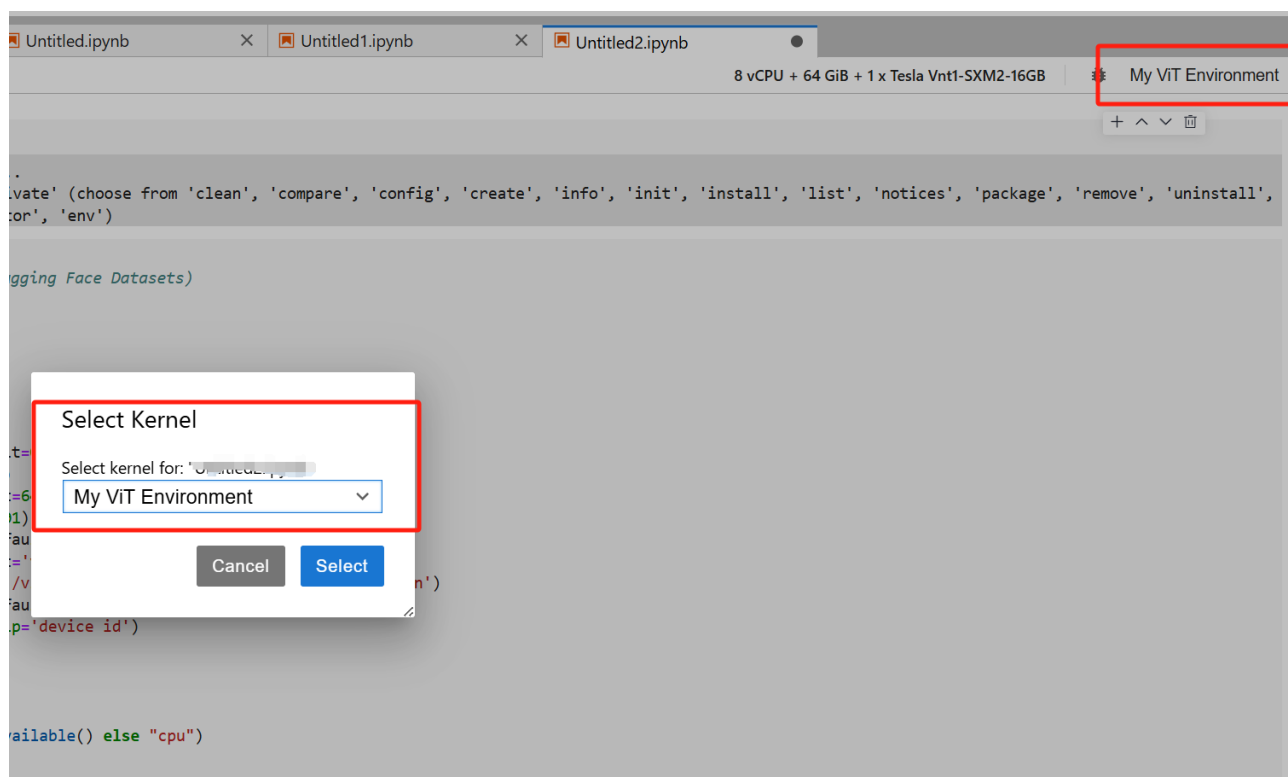
```
conda create -n vit_env python=3.10 -y
conda activate vit_env
mv ~/modelarts-dev/modelarts-sdk/pandas ~/modelarts-dev/modelarts-sdk/pandas_backup
conda activate vit_env
pip install torch==2.5.1 torchvision==0.20.1 torchaudio --index-url https://download.pytorch.org/whl/cu118
pip install datasets # 这里往往会自动给你安装 pandas 和 pyarrow
pip install pandas tqdm # 如果还有其它库，统一装上
pip install ipykernel
python -m ipykernel install --user --name=vit_env --display-name "My ViT Environment"
```

```
export HF_ENDPOINT="https://hf-mirror.com"
```

```
Requirement already satisfied: filelock in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (3.17.0)
Requirement already satisfied: typing-extensions==4.8.0 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (4.12.2)
Requirement already satisfied: networkx in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (3.2.1)
Requirement already satisfied: jinja2 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (3.1.3)
Requirement already satisfied: fsspec in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (2024.9.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu11==11.8.89 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (11.8.89)
Requirement already satisfied: nvidia-cuda-runtime-cu11==11.8.89 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (11.8.89)
Requirement already satisfied: nvidia-cuda-cupti-cu11==11.8.89 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (11.8.89)
Requirement already satisfied: nvidia-cublas-cu11==9.1.0.70 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu11==11.11.3.6 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (11.11.3.6)
Requirement already satisfied: nvidia-cufft-cu11==10.9.0.58 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (10.9.0.58)
Requirement already satisfied: nvidia-curand-cu11==10.3.0.86 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (10.3.0.86)
Requirement already satisfied: nvidia-cusolver-cu11==11.4.1.48 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (11.4.1.48)
Requirement already satisfied: nvidia-cuspars-cu11==11.7.5.86 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (11.7.5.86)
Requirement already satisfied: nvidia-nccl-cu11==2.12.5 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (2.12.5)
Requirement already satisfied: nvidia-nvtx-cu11==11.8.86 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (11.8.86)
Requirement already satisfied: triton==3.1.0 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (3.1.0)
Requirement already satisfied: sympy==1.13.1 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torch==2.5.1) (1.13.1)
Requirement already satisfied: numpy in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torchvision==0.20.1) (2.2.2)
Requirement already satisfied: pillow==8.3.4>=5.3.0 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torchvision==0.20.1) (10.2.0)
Requirement already satisfied: imageio==2.34.0 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from torchvision==0.20.1) (2.34.0)
Requirement already satisfied: MarkupSafe>=2.0 in /home/ma-user/anaconda3/envs/vit_env/lib/python3.10/site-packages (from jinja2->torch==2.5.1) (2.1.5)
Installing collected packages: torch, torchvision, torchaudio
Successfully installed torch-2.5.1+cu118 torchaudio-2.5.1+cu118 torchvision-0.20.1+cu118
(vit_env) [ma-user@work]$
```

步骤 2 激活对应的环境

开启 notebook 后，请检查是否是使用 My Vit Enviroment



步骤 3 定义变量

代码：

```
import os
os.environ["HF_ENDPOINT"] = "https://hf-mirror.com"
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('--num_classes', type=int, default=0) # 先默认 0，后面自动检测
parser.add_argument('--epochs', type=int, default=20)
#建议设置为 50 次，效果更好。
parser.add_argument('--batch_size', type=int, default=64)
parser.add_argument('--lr', type=float, default=0.0001)
```

```
parser.add_argument('--weight_decay', type=float, default=0.01)
parser.add_argument('--model_name', type=str, default='vit_tiny')
parser.add_argument('--weights', type=str, default='./vit-tiny-patch16-224/pytorch_model.bin')
#可以调节到 vit base，反正模型也下了。
parser.add_argument('--freeze_layers', type=bool, default=False)
#如果想训练整个模型，可以设置为 False
parser.add_argument('--device', default='cuda:0', help='device id')

args = parser.parse_args([]) # 兼容 Notebook

import torch
device = torch.device(args.device if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

1.2.3 数据处理与加载

使用 RandomCropDecodeResize、RandomHorizontalFlip、Normalize 等进行数据预处理。

代码：

```
###
!pip install -q datasets # 若已安装可注释掉
from datasets import load_dataset

# 假设你的数据集名为 "some-user/tiny-imagenet"
# 通常会有 train / validation 两个分割
dataset = load_dataset("israelsr/mm_tiny_imagenet")

print(dataset) # 查看基本信息, e.g. DatasetDict with "train", "validation"

# 这里可以自动获取类别数 (若数据集中包含 label 列)
# 大多数图像数据集的 label 会是 ClassLabel 类型, 提供 num_classes
args.num_classes = dataset["train"].features["label"].num_classes
print(f"实际数据集类别数: {args.num_classes}")

import torchvision.transforms as T
from PIL import Image

IMAGENET_MEAN = [0.485, 0.456, 0.406]
IMAGENET_STD = [0.229, 0.224, 0.225]

# 把一张 PIL Image 无论是灰度还是 RGBA, 都转成 RGB
def pil_to_rgb(img: Image.Image) -> Image.Image:
    return img.convert("RGB")

train_transforms = T.Compose([
    T.Lambda(pil_to_rgb), # 强制转 RGB
    T.RandomResizedCrop(224, scale=(0.08, 1.0), ratio=(0.75, 1.33)),
    T.RandomHorizontalFlip(),
```

```
T.ToTensor(),
T.Normalize(IMAGENET_MEAN, IMAGENET_STD),
])

val_transforms = T.Compose([
    T.Lambda(pil_to_rgb),          # 强制转 RGB
    T.RandomResizedCrop(224, scale=(0.08, 1.0), ratio=(0.75, 1.33)),
    T.RandomHorizontalFlip(),
    T.ToTensor(),
T.Normalize(IMAGENET_MEAN, IMAGENET_STD),
])

def train_transform_fn(examples):
    # examples 是一个批量, 比如{"image": [...], "label": [...]}
    # 对每张图做 transforms
    images = [train_transforms(img) for img in examples["image"]]
    labels = examples["label"]
    return {"pixel_values": images, "labels": labels}

def val_transform_fn(examples):
    images = [val_transforms(img) for img in examples["image"]]
    labels = examples["label"]
    return {"pixel_values": images, "labels": labels}

# 给 train / validation 集分别 set_transform
dataset["train"].set_transform(train_transform_fn)
dataset["validation"].set_transform(val_transform_fn)

### 2.2 转成 PyTorch DataLoader

%%
from torch.utils.data import DataLoader

train_loader = DataLoader(
    dataset["train"],
    batch_size=128,
    shuffle=True,
    num_workers=6,
    pin_memory=True,
    persistent_workers=True
)

val_loader = DataLoader(
    dataset["validation"],
    batch_size=128,
    shuffle=False,
    num_workers=6,
```

```
pin_memory=True,
persistent_workers=True
)

# 小测试: 取一个批次
first_batch = next(iter(train_loader))
print("Batch keys:", first_batch.keys())
print("pixel_values shape:", first_batch["pixel_values"].shape)
print("labels shape:", first_batch["labels"].shape)
```

1.2.4 模型构建

步骤 1 定义 Patch Embedding 层，Feed Forward 层。

在代码实现中，直接通过一个卷积层来实现。以 ViT-B/16 为例，直接使用一个卷积核大小为 16x16，步距为 16，卷积核个数为 768 的卷积来实现。通过卷积[224, 224, 3] -> [14, 14, 768]，然后把 H 以及 W 两个维度展平即可[14, 14, 768] -> [196, 768]，此时正好变成了一个二维矩阵，正是 Transformer Encoder 想要的。这里还需要加上一个特殊字符 cls，因此最终的维度是 197x768。

ViT 同样需要加入位置编码，位置编码可以理解为一张表，表一共有 N 行，N 的大小和输入序列长度相同，每一行代表一个向量，向量的维度和输入序列 embedding 的维度相同（768）。注意位置编码的操作是 sum，而不是 concat。加入位置编码信息之后，维度依然是 197x768。

代码：

```
import math
import torch.nn as nn
from torch.nn.init import _no_grad_normal_

def trunc_normal_(tensor, mean=0., std=1., a=-2., b=2.):
    _no_grad_normal_(tensor, mean=mean, std=std)
    with torch.no_grad():
        tensor.clamp_(min=a*std, max=b*std)
    return tensor

class PatchEmbedding(nn.Module):
    def __init__(self, img_size=224, patch_size=16, in_chans=3, embed_dim=768):
        super().__init__()
        self.proj = nn.Conv2d(in_chans, embed_dim,
                               kernel_size=patch_size, stride=patch_size)
        self.norm = nn.LayerNorm(embed_dim)

        trunc_normal_(self.proj.weight, std=0.02)
```

```
nn.init.zeros_(self.proj.bias)

def forward(self, x):
    x = self.proj(x)      # [B, embed_dim, H/patch, W/patch]
    x = x.flatten(2)      # [B, embed_dim, num_patches]
    x = x.transpose(1, 2) # [B, num_patches, embed_dim]
    return self.norm(x)
```

步骤 2 定义 Residual Connection 结构

在了解了 Self-Attention 结构之后，通过与 Feed Forward，Residual Connection 等结构的拼接就可以形成 Transformer 的基础结构。

下面代码实现了 Feed Forward，Residual Connection 结构。

```
import torch
import torch.nn as nn
from typing import Optional

class Residual(nn.Module):
    def __init__(self, fn):
        super(Residual, self).__init__()
        self.fn = fn

    def forward(self, x):
        return self.fn(x) + x
```

步骤 3 定义 ViT

接下来将上述结构组合成一个 Vision Transformer Encoder Block。

ViT 模型中的基础结构与标准 Transformer 有所不同，主要在于 Normalization 的位置是放在 Self-Attention 和 Feed Forward 之前，其他结构如 Residual Connection，Feed Forward，Normalization 都如 Transformer 中所设计。

Transformer 中，多个子 encoder 的堆叠就完成了模型编码器的构建，在 ViT 模型中，依然沿用这个思路，通过配置超参数 num_layers，就可以确定堆叠层数。

Residual Connection，Normalization 的结构可以保证模型有很强的扩展性（保证信息经过深层处理不会出现退化的现象，这是 Residual Connection 的作用），Normalization 和 dropout 的应用可以增强模型泛化能力。

代码：

```
class ViT(nn.Module):
    def __init__(self,
                 num_classes=1000,
                 img_size=224,
                 patch_size=16,
                 in_chans=3,
```

```
        embed_dim=768,
        depth=12,
        num_heads=12,
        mlp_ratio=4.0,
        dropout=0.1):
    super().__init__()

    self.patch_embed = PatchEmbedding(img_size, patch_size, in_chans, embed_dim)
    num_patches = (img_size // patch_size) * (img_size // patch_size)

    self.cls_token = nn.Parameter(torch.zeros(1, 1, embed_dim))
    self.pos_embed = nn.Parameter(torch.zeros(1, num_patches + 1, embed_dim))
    trunc_normal_(self.cls_token, std=0.02)
    trunc_normal_(self.pos_embed, std=0.02)

    # PyTorch 1.8 不支持 batch_first=True / norm_first=True
    # 这里手动做 [B,N,C]=>[N,B,C] 再进 TransformerEncoder
    encoder_layer = nn.TransformerEncoderLayer(
        d_model=embed_dim,
        nhead=num_heads,
        dim_feedforward=int(embed_dim * mlp_ratio),
        dropout=dropout,
        activation='gelu'
    )
    self.encoder = nn.TransformerEncoder(encoder_layer, num_layers=depth)

    self.norm = nn.LayerNorm(embed_dim)
    self.head = nn.Linear(embed_dim, num_classes)

    nn.init.zeros_(self.head.weight)
    nn.init.zeros_(self.head.bias)

    def forward(self, x):
        B = x.shape[0]
        x = self.patch_embed(x) # [B, num_patches, embed_dim]

        cls_tokens = self.cls_token.expand(B, -1, -1) # [B, 1, embed_dim]
        x = torch.cat((cls_tokens, x), dim=1) # [B, 1+num_patches, embed_dim]

        x = x + self.pos_embed[:, : x.shape[1], :] # 广播加法

        # => [N, B, C]
        x = x.transpose(0, 1)
        x = self.encoder(x)
        x = x.transpose(0, 1) # => [B, N, C]

        x = self.norm(x) # [B, N, C]
```

```
cls_x = x[:, 0]          # [B, embed_dim]
return self.head(cls_x)
```

1.2.5 模型训练和评估

步骤 1 加载模型与定义超参数

代码：

```
# define super parameter
model = ViT(
    num_classes=args.num_classes,
    img_size=224,
    patch_size=16,
    embed_dim=768,
    depth=12,
    num_heads=12,
    mlp_ratio=4.0,
    dropout=0.1
).to(device)

# 加载预训练
pretrained_dict = torch.load(args.weights, map_location=device)
if isinstance(pretrained_dict, dict) and 'model' in pretrained_dict:
    pretrained_dict = pretrained_dict['model']

model_dict = model.state_dict()
filtered_dict = {}
for k, v in pretrained_dict.items():
    if k in model_dict and 'head' not in k:
        filtered_dict[k] = v

model_dict.update(filtered_dict)
model.load_state_dict(model_dict, strict=False)
print("=> Pretrained weights loaded with partial matching.")

if args.freeze_layers:
    for name, param in model.named_parameters():
        if 'head' not in name:
            param.requires_grad_(False)
    print("=> Model backbone layers have been frozen.")
```

步骤 2 完成训练等任务

代码：

```
import torch
```

```
from torch.cuda.amp import autocast, GradScaler
from tqdm import tqdm

scaler = GradScaler()

params = [
    {"params": [p for n, p in model.named_parameters() if "head" in n], "lr": args.lr * 10},
    {"params": [p for n, p in model.named_parameters() if "head" not in n], "lr": args.lr}
]

optimizer = torch.optim.AdamW(params, weight_decay=args.weight_decay)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=args.epochs)
criterion = nn.CrossEntropyLoss().to(device)

best_acc = 0.0

for epoch in range(args.epochs):
    model.train()
    train_loss = 0.0
    correct_train = 0
    total_train = 0

    pbar = tqdm(train_loader, desc=f'Epoch [{epoch+1}/{args.epochs}]')
    for batch in pbar:
        # 训练代码
        images = batch["pixel_values"].to(device, non_blocking=True)
        labels = batch["labels"].to(device, non_blocking=True)

        optimizer.zero_grad(set_to_none=True)

        with autocast():
            outputs = model(images)
            loss = criterion(outputs, labels)

        scaler.scale(loss).backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=5.0)
        scaler.step(optimizer)
        scaler.update()

        train_loss += loss.item() * images.size(0)
        _, preds = outputs.max(dim=1)
        correct_train += (preds == labels).sum().item()
        total_train += labels.size(0)

    train_epoch_loss = train_loss / total_train
    train_acc = correct_train / total_train
```

```
# 验证
model.eval()
correct_val = 0
total_val = 0

with torch.no_grad():
    for batch in val_loader: # 使用预先定义的 val_loader
        images = batch["pixel_values"].to(device, non_blocking=True)
        labels = batch["labels"].to(device, non_blocking=True)
        outputs = model(images)
        preds = outputs.argmax(dim=1)
        correct_val += (preds == labels).sum().item()
        total_val += labels.size(0)

val_acc = correct_val / total_val
scheduler.step()

print(f"Epoch [{epoch+1}/{args.epochs}] "
      f"Train Loss: {train_epoch_loss:.4f} | "
      f"Train Acc: {train_acc:.2%} | "
      f"Val Acc: {val_acc:.2%}")

if val_acc > best_acc:
    best_acc = val_acc
    torch.save(model.state_dict(), "best_vit.pth")
    print(f"*** Best model saved with val acc: {best_acc:.2%} ***")

print(f"Training Completed. Best val acc = {best_acc:.2%}")
```

输出：

```
Epoch [1/20]: 100% ██████████ | 625/625 [03:58<00:00, 2.62it/s]
Epoch [1/20] Train Loss: 5.0720 | Train Acc: 2.25% | Val Acc: 3.81%
** Best model saved with val acc: 3.81% **
Epoch [2/20]: 100% ██████████ | 625/625 [03:56<00:00, 2.64it/s]
Epoch [2/20] Train Loss: 4.6924 | Train Acc: 5.61% | Val Acc: 7.43%
** Best model saved with val acc: 7.43% **
Epoch [3/20]: 100% ██████████ | 625/625 [03:57<00:00, 2.63it/s]
Epoch [3/20] Train Loss: 4.4090 | Train Acc: 8.52% | Val Acc: 9.88%
** Best model saved with val acc: 9.88% **
Epoch [4/20]: 100% ██████████ | 625/625 [03:57<00:00, 2.63it/s]
Epoch [4/20] Train Loss: 4.1050 | Train Acc: 12.78% | Val Acc: 14.30%
** Best model saved with val acc: 14.30% **
Epoch [5/20]: 100% ██████████ | 625/625 [03:57<00:00, 2.63it/s]
Epoch [5/20] Train Loss: 3.9343 | Train Acc: 15.14% | Val Acc: 16.24%
** Best model saved with val acc: 16.24%
```

这里的训练速度会比较慢，截取了部分输出作为参考变化，可以看到随着 Epoch 上升，模型能力出现上升，可以适当调节 Epoch 以更好地拟合。

1.2.6 模型验证

与训练过程相似，首先进行数据增强，然后定义 ViT 网络结构，加载预训练模型参数。随后设置损失函数，评价指标等，编译模型后进行验证。本实验采用了业界通用的评价标准 Top_1_Accuracy 和 Top_5_Accuracy 评价指标来评价模型表现。这两个指标代表了在输出的 1000 维向量中，以最大值或前 5 的输出值所代表的类别为预测结果时，模型预测的准确率。这两个指标的值越大，代表模型准确率越高。

代码：

```
def evaluate(model, val_loader, criterion, device):
    model.eval()
    running_loss = 0.0
    top1_correct = 0
    top5_correct = 0
    total = 0

    with torch.no_grad():
        for batch in val_loader:
            images = batch["pixel_values"].to(device, non_blocking=True)
            labels = batch["labels"].to(device, non_blocking=True)

            outputs = model(images)
            loss = criterion(outputs, labels)

            # Top-1 和 Top-5 计算
            _, preds = outputs.topk(5, dim=1)
            correct = preds.eq(labels.view(-1, 1)) # 检查预测是否正确

            top1_correct += correct[:, 0].sum().item() # Top-1 正确个数
            top5_correct += correct.any(dim=1).sum().item() # Top-5 正确个数
            total += labels.size(0)

            running_loss += loss.item()

    val_loss = running_loss / len(val_loader)
    top1_acc = top1_correct / total
    top5_acc = top5_correct / total

    return val_loss, top1_acc, top5_acc
```

```
# 在验证阶段调用 evaluate
val_loss, top1_acc, top5_acc = evaluate(
    model=model,
    val_loader=val_loader,
    criterion=criterion,
    device=device
)

print(f"Validation Loss: {val_loss:.4f}")
print(f"Top-1 Accuracy: {top1_acc:.2%}")
print(f"Top-5 Accuracy: {top5_acc:.2%}")
```

输出：

```
Validation Loss: 1.19
Top-1 Accuracy: 79.13%
Top-5 Accuracy 91.33%
#(epoch 为 50 轮)
```

1.2.7 模型推理

在进行模型推理之前，首先要定义一个对推理图片进行数据预处理的方法。该方法可以对推理图片进行 resize 和 normalize 处理。本实验采用了一张图片作为推理图片来测试模型表现。

代码：

```
from PIL import Image
import torch
import torchvision.transforms as T
import torchvision.transforms.functional as F

# 定义反归一化操作
def denormalize(tensor, mean, std):
    """
    反归一化图像张量，将像素值恢复到原始范围。
    """
    for t, m, s in zip(tensor, mean, std):
        t.mul_(s).add_(m) # 逆操作：乘标准差，加均值
    return tensor

def infer_from_validation(val_dataset, model, device='cuda', idx=0):
    """
    从验证集抽取样本进行推理，并展示推理结果和图片。
    """
    # 数据预处理
    transform = T.Compose([
        T.Resize(224)
    ])
```

```
# 从验证集获取图像和真实标签
example = val_dataset[idx]
image_tensor = example["pixel_values"] # 已是 Tensor
label = example["labels"]

# 如果图像是 Tensor, 先转回 PIL.Image 格式用于显示
if isinstance(image_tensor, torch.Tensor):
    # 转换为 [0, 1] 范围
    original_image = denormalize(image_tensor.clone(), mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    original_image = torch.clamp(original_image, 0, 1) # 保证像素值范围在 [0, 1]
    image_pil = F.to_pil_image(original_image) # 转回 PIL.Image
else:
    raise TypeError(f'Expected image to be a Tensor, but got {type(image_tensor)}')

# 将图像 Tensor 移动到设备并应用归一化预处理
image_tensor = image_tensor.to(device)
image_tensor = transform(image_tensor.unsqueeze(0)) # 添加 batch 维度并应用变换

# 推理
model.eval()
with torch.no_grad():
    output = model(image_tensor)
    prob = torch.softmax(output, dim=1)
    label_idx = torch.argmax(prob, dim=1).item()

# 显示推理结果
print(f'Predicted: {label_idx} (Prob: {prob[0][label_idx]:.2f}), True: {label}')

# 使用 PIL.Image 显示图像
image_pil.show(title=f'Predicted: {label_idx} (Prob: {prob[0][label_idx]:.2f})\nTrue: {label}')

infer_from_validation(
    val_dataset=dataset["validation"], # 验证集
    model=model,                       # 已训练的模型
    device=device,                     # 推理设备
    idx=2238                           # 抽取验证集中第 2238 个样本
)
```

输出：

Predicted: 11 (Prob: 0.63), True:11



1.3 实验小节

本章提供了一个基于 Pytorch 框架的 Vision Transformer 花卉图像识别实验。该实验演示了如何利用 Pytorch 搭建网络构建 ViT 模型，再利用华为云 ModelArts 提供的 GPU 算力，完成训练和评估，实现图像识别。

1.4 思考题

在上述实验中的推理函数中，可以使用训练集放入进行操作吗？

答：可以，将 validation 改为 train 即可。