

华为认证 AI 系列教程

HCIA-AI

机器学习实验指导手册

版本：3.5



华为技术有限公司

版权所有 © 华为技术有限公司 2022。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技术架构与应用认证、云服务与平台认证两类认证。

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。

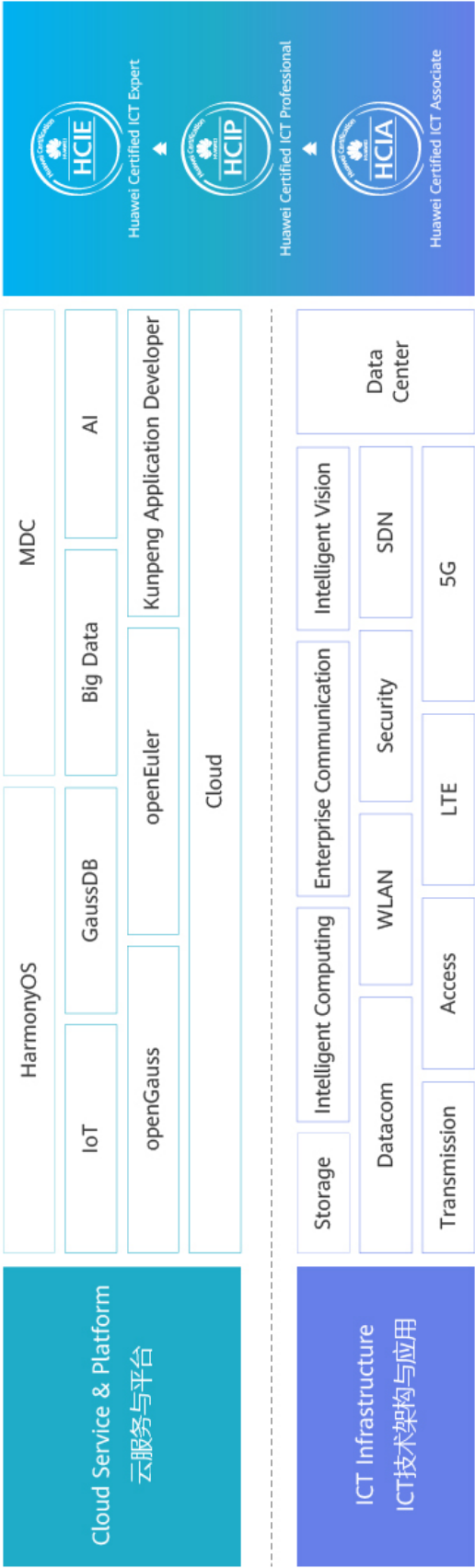
华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIA-AI V3.5定位于培养和认证具备使用机器学习、深度学习等算法进行AI产品和解决方案设计、开发和创新能力的工程师。

通过HCIA-AI V3.5认证，将证明您了解人工智能发展历史、华为昇腾AI体系和全栈全场景AI战略知识，掌握传统机器学习和深度学习的相关算法；具备利用MindSpore开发框架和MindSpore开发框架进行搭建、训练、部署神经网络的能力；能够胜任人工智能领域销售、市场、产品经理、项目管理、技术支持等岗位。

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！

华为认证



简介

本书为 HCIA-AI 认证培训教程，适用于准备参加 HCIA-AI 考试的学员或者希望了解机器学习基础知识的读者。

内容描述

本实验指导书共包含 6 个小的实验。

- 实验一线性回归，通过 Python 工具包 scikit-learn 实现简单线性回归算法的使用。
- 实验二为线性回归拓展实验，通过 Python 基础工具包 numpy 等从零实现线性回归、梯度下降等算法，加深理解。
- 实验三逻辑回归，通过工具包中的逻辑回归算法实现简单的分类任务。
- 实验四为决策树算法的使用，通过构建决策树实现天气预测，并可视化决策树算法加深理解。
- 实验五为垃圾邮件分类，通过朴素贝叶斯算法实现垃圾邮件分类。
- 实验六为聚类算法 K-means 的使用。

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 Python 编程基础知识；
- 具备线性代数、概率论等基础数学知识。

实验环境说明

本实验环境为 Python3.7（相关环境搭建，请参考环境搭建指南）。

实验数据链接：<https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V3.5/chapter2/ML.zip>

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 常见机器学习算法实现	5
1.1 实验介绍	5
1.1.1 关于本实验	5
1.1.2 实验目的	5
1.2 代码实现	5
1.2.1 线性回归	5
1.2.2 线性回归算法实现（扩展）	7
1.2.3 逻辑回归	10
1.2.4 决策树	12
1.2.5 朴素贝叶斯	14
1.2.6 K-means 算法实现	17
1.3 思考题	22
1.4 本章总结	22

1 常见机器学习算法实现

1.1 实验介绍

1.1.1 关于本实验

本实验包含了常见机器学习算法的使用，通过从零开始构建线性回归算法和基于 scikit-learn 调用实现决策树、朴素贝叶斯、聚类算法，帮助您更加了解机器学习算法的使用流程和功能。

1.1.2 实验目的

- 实现从零开始构建线性回归算法。
- 掌握分类、回归算法的使用。
- 掌握 V 聚类算法的使用。
- 掌握机器学习应用实现流程。

1.2 代码实现

1.2.1 线性回归

步骤 1 引入相关依赖的包

输入：

```
from sklearn.linear_model import LinearRegression#导入线性回归模型
import matplotlib.pyplot as plt#绘图库
import numpy as np
```

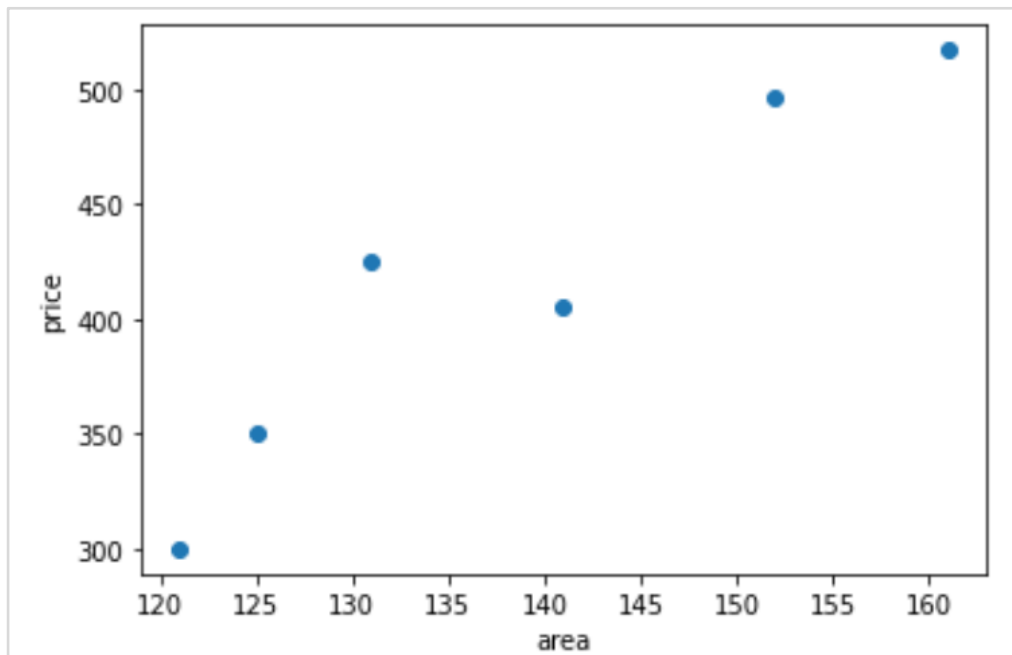
步骤 2 构建房价数据集并可视化

输入：

```
x = np.array([121, 125, 131, 141, 152, 161]).reshape(-1,1)#x 是房屋面积，作为特征
y = np.array([300, 350, 425, 405,496,517])#y 是房屋的
plt.scatter(x,y)
plt.xlabel("area")#添加横坐标面积
plt.ylabel("price")#添加纵坐标价格
```

```
plt.show()
```

输出：



步骤 3 模型训练

输入：

```
lr = LinearRegression()#将线性回归模型封装为对象
lr.fit(x,y)#模型在数据上训练
```

步骤 4 模型的可视化

输入：

```
w = lr.coef_#存储模型的斜率
b = lr.intercept_#存储模型的截距
print('斜率:',w)
print('截距:',b)
```

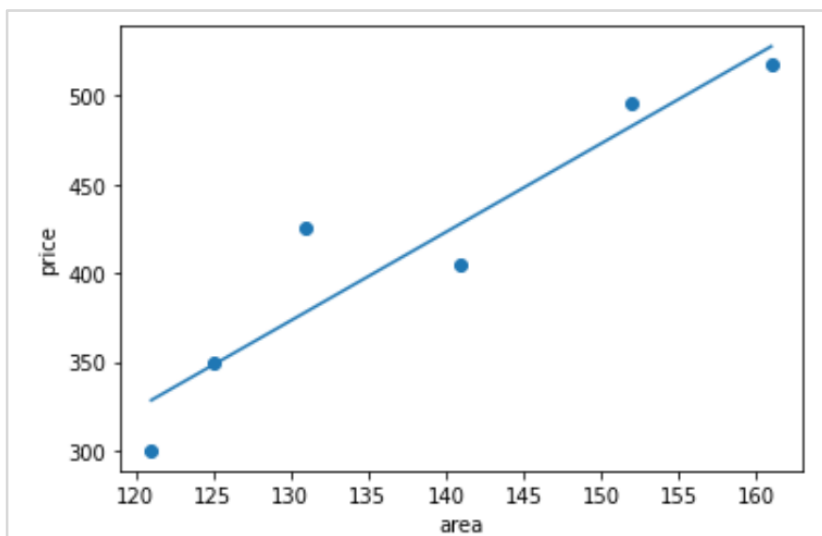
输出：

```
斜率: [4.98467124]
截距: -274.8769665187576
```

输入：

```
plt.scatter(x,y)
plt.xlabel("area")#添加横坐标面积
plt.ylabel("price")#添加纵坐标价格
plt.plot([x[0],x[-1]], [x[0]*w+b,x[-1]*w+b])
```

输出：



步骤 5 模型预测

输入：

```
testX = np.array([[130]])#测试样本，面积为 130
lr.predict(testX)
```

输出：

```
array([373.13029447])#模型预测获得样本的房价
```

1.2.2 线性回归算法实现（扩展）

在“线性回归算法实现”部分，使用的数据集为 lr2_data.txt，主要是自行模拟的房屋面积与房屋价格数据。该数据集可在实验环境搭建中获取。

步骤 1 导入依赖

输入：

```
import numpy as np
import matplotlib.pyplot as plt
```

步骤 2 定义函数，计算梯度

输入：

```
def generate_gradient(X, theta, y):
    sample_count = X.shape[0]
    # 计算梯度，采用矩阵计算  $1/m \sum ((h(x^i) - y^i)) x_j^i$ 
    return (1./sample_count)*X.T.dot(X.dot(theta)-y)
```

步骤 3 定义函数，用于读取数据集

输入：

```
def get_training_data(file_path):
```

```
orig_data = np.loadtxt(file_path, skiprows=1) #忽略第一行的标题
cols = orig_data.shape[1]
return (orig_data, orig_data[:, :cols - 1], orig_data[:, cols-1:])
```

步骤 4 定义函数，初始化参数

输入：

```
# 初始化  $\theta$  数组
def init_theta(feature_count):
    return np.ones(feature_count).reshape(feature_count, 1)
```

步骤 5 定义函数，实现梯度下降法

输入：

```
def gradient_descending(X, y, theta, alpha):
    Jthetas = [] # 记录代价函数  $J(\theta)$  的变化趋势，验证梯度下降是否运行正确
    # 计算损失函数，等于真实值与预测值差的平方。 $(y^i - h(x^i))^2$ 
    Jtheta = (X.dot(theta) - y).T.dot(X.dot(theta) - y)
    index = 0
    gradient = generate_gradient(X, theta, y) # 计算梯度
    while not np.all(np.absolute(gradient) <= 1e-5): # 梯度小于 0.00001 时计算结束
        theta = theta - alpha * gradient
        gradient = generate_gradient(X, theta, y) # 计算新梯度
        # 计算损失函数，等于真实值与预测值差的平方  $(y^i - h(x^i))^2$ 
        Jtheta = (X.dot(theta) - y).T.dot(X.dot(theta) - y)
        if (index+1) % 10 == 0:
            Jthetas.append((index, Jtheta[0])) # 每 10 次计算记录一次结果
        index += 1
    return theta, Jthetas
```

步骤 6 定义函数，可视化损失函数变化曲线

输入：

```
# 展示损失函数变化曲线图
def showJTheta(diff_value):
    p_x = []
    p_y = []
    for (index, sum) in diff_value:
        p_x.append(index)
        p_y.append(sum)
    plt.plot(p_x, p_y, color='b')
    plt.xlabel('steps')
    plt.ylabel('loss function')
    plt.title('step - loss function curve')
    plt.show()
```

步骤 7 定义函数，可视化数据点及拟合的曲线

输入：

```
#展示实际数据点以及拟合出的曲线图
def showlinercurve(theta, sample_training_set):
    x, y = sample_training_set[:, 1], sample_training_set[:, 2]
    z = theta[0] + theta[1] * x
    plt.scatter(x, y, color='b', marker='x', label="sample data")
    plt.plot(x, z, 'r', color="r", label="regression curve")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('liner regression curve')
    plt.legend()
    plt.show()
```

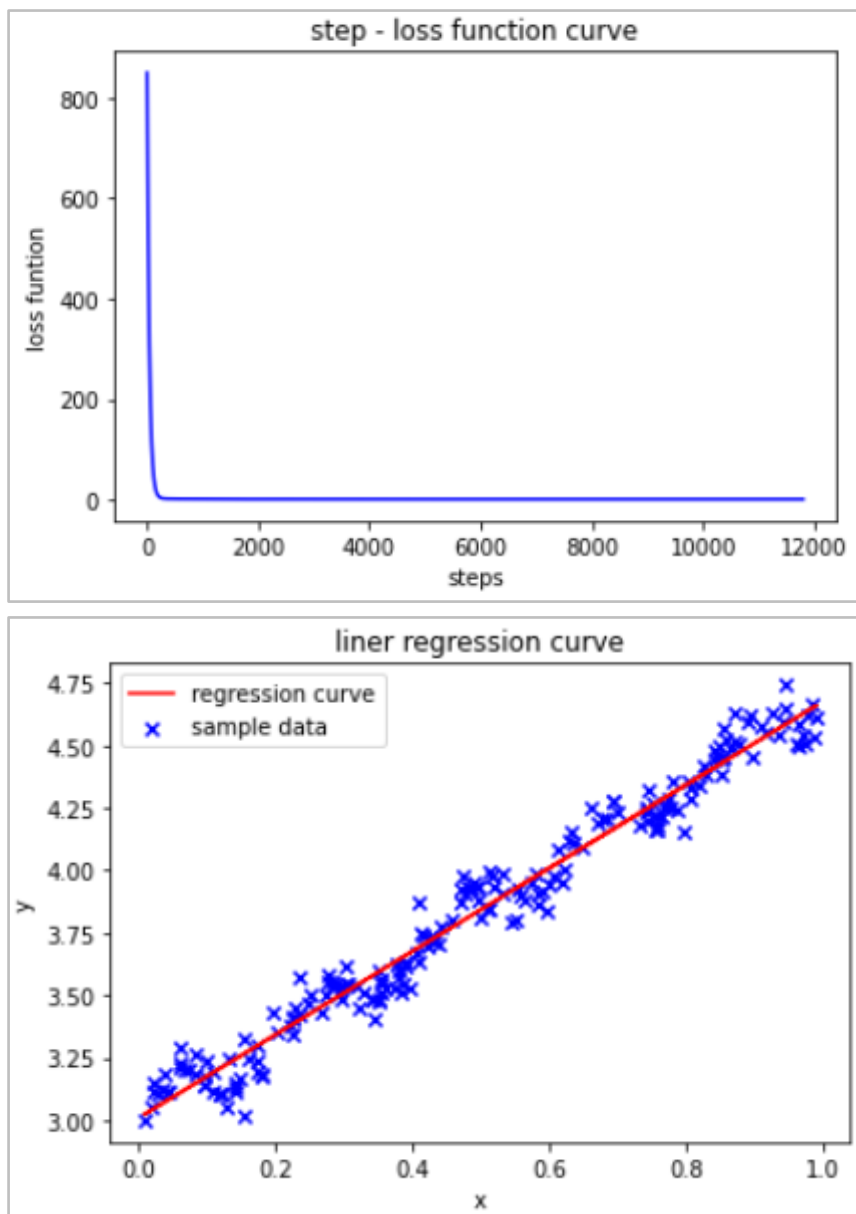
步骤 8 最终展示

输入：

```
# 读取数据集
training_data_include_y, training_x, y = get_training_data("./ML/02/lr2_data.txt")
# 获取数据集数量及特征数
sample_count, feature_count = training_x.shape
# 定义学习步长  $\alpha$ 
alpha = 0.01
# 初始化  $\theta$ 
theta = init_theta(feature_count)
# 获取最终的参数  $\theta$  及代价
result_theta, Jthetas = gradient_descending(training_x, y, theta, alpha)
# 打印参数
print("w:{}".format(result_theta[0][0]), "b:{}".format(result_theta[1][0]))
showJTheta(Jthetas)
showlinercurve(result_theta, training_data_include_y)
```

输出（warning 可以忽略）：

```
w:3.0076279423997594 b:1.668677412281192
```



验证 VLAN 的二层隔离特性。

1.2.3 逻辑回归

在逻辑回归部分，使用的数据集为自定义的房屋租金和面积相关的数据集，在实验初始阶段会进行定义。

步骤 1 导入依赖

输入：

```
# 从 sklearn.preprocessing 里导入 StandardScaler。
from sklearn.preprocessing import StandardScaler
# 从 sklearn.linear_model 里导入 LogisticRegression
from sklearn.linear_model import LogisticRegression
```

步骤 2 自定义数据集

输入：

```
# X: 每一项表示租金和面积
# y: 表示是否租赁该房间 (0: 不租, 1: 租)
X=[[2200,15],[2750,20],[5000,40],[4000,20],[3300,20],[2000,10],[2500,12],[12000,80],
   [2880,10],[2300,15],[1500,10],[3000,8],[2000,14],[2000,10],[2150,8],[3400,20],
   [5000,20],[4000,10],[3300,15],[2000,12],[2500,14],[10000,100],[3150,10],
   [2950,15],[1500,5],[3000,18],[8000,12],[2220,14],[6000,100],[3050,10]
  ]

y=[1,1,0,0,1,1,1,1,0,1,1,0,1,1,0,1,0,0,0,1,1,1,0,1,0,1,0,1,1,0]
```

步骤 3 数据预处理

标准化数据，保证每个维度的特征数据方差为 1，均值为 0。使得预测结果不会被某些维度过大的特征值而主导。

输入：

```
ss = StandardScaler()
X_train = ss.fit_transform(X)
```

查看标准化处理后的数据。

输入：

```
print(X_train)
```

输出：

```
[[-0.60583897 -0.29313058]
 [-0.37682768 -0.09050576]
 [ 0.56003671  0.71999355]
 [ 0.14365254 -0.09050576]
 [-0.14781638 -0.09050576]
 [-0.68911581 -0.49575541]
 [-0.48092372 -0.41470548]
 [ 3.47472592  2.34099218]
 [-0.32269773 -0.49575541]
 [-0.56420055 -0.29313058]
 [-0.89730789 -0.49575541]
 [-0.27273163 -0.57680534]
 [-0.68911581 -0.33365555]
 [-0.68911581 -0.49575541]
 [-0.62665818 -0.57680534]
 [-0.10617796 -0.09050576]
 [ 0.56003671 -0.09050576]
 [ 0.14365254 -0.49575541]
 [-0.14781638 -0.29313058]
 [-0.68911581 -0.41470548]
 [-0.48092372 -0.33365555]]
```

```
[ 2.64195758  3.15149149]
[-0.21027401 -0.49575541]
[-0.29355084 -0.29313058]
[-0.89730789 -0.69838024]
[-0.27273163 -0.17155569]
[ 1.80918923 -0.41470548]
[-0.59751129 -0.33365555]
[ 0.97642089  3.15149149]
[-0.25191242 -0.49575541]]
```

步骤 4 数据拟合

输入：

```
#调用 Lr 中的 fit 模块训练模型参数
lr = LogisticRegression()
lr.fit(X_train, y)
```

输出：

```
LogisticRegression()
```

步骤 5 数据预测

输入：

```
testX = [[2000,8]]
X_test = ss.transform(testX)
print("待预测的值：",X_test)
label = lr.predict(X_test)
print("predicted label = ", label)
#输出预测概率
prob = lr.predict_proba(X_test)
print("probability = ",prob)
```

输出：

```
待预测的值： [[-0.68911581 -0.57680534]]
predicted label = [1]
probability = [[0.41886952 0.58113048]]
```

1.2.4 决策树

本实验使用的数据集为 tennis.txt，主要包含 14 个样本，每个样本包含天气相关的特征及是否适合打球。

步骤 1 导入依赖

输入：

```
import pandas as pd
import numpy as np
from sklearn import tree
```

```
import pydotplus
```

步骤 2 定义函数，生成决策树

输入：

```
#生成决策树
def createTree(trainingData):
    data = trainingData.iloc[:, :-1] # 特征矩阵
    labels = trainingData.iloc[:, -1] # 标签
    trainedTree = tree.DecisionTreeClassifier(criterion="entropy") # 分类决策树
    trainedTree.fit(data, labels) # 训练
    return trainedTree
```

步骤 3 定义函数，保存生成的树图

输入：

```
def showtree2pdf(trainedTree, filename):
    dot_data = tree.export_graphviz(trainedTree, out_file=None) #将树导出为 Graphviz 格式
    graph = pydotplus.graph_from_dot_data(dot_data)
    graph.write_pdf(filename) #保存树图到本地，格式为 pdf
```

步骤 4 定义函数，用于生成向量化数据

函数中，通过 `pd.Categorical(list).codes` 可以得到原始数据对应的序号列表，从而将类别信息转化成数值信息。

输入：

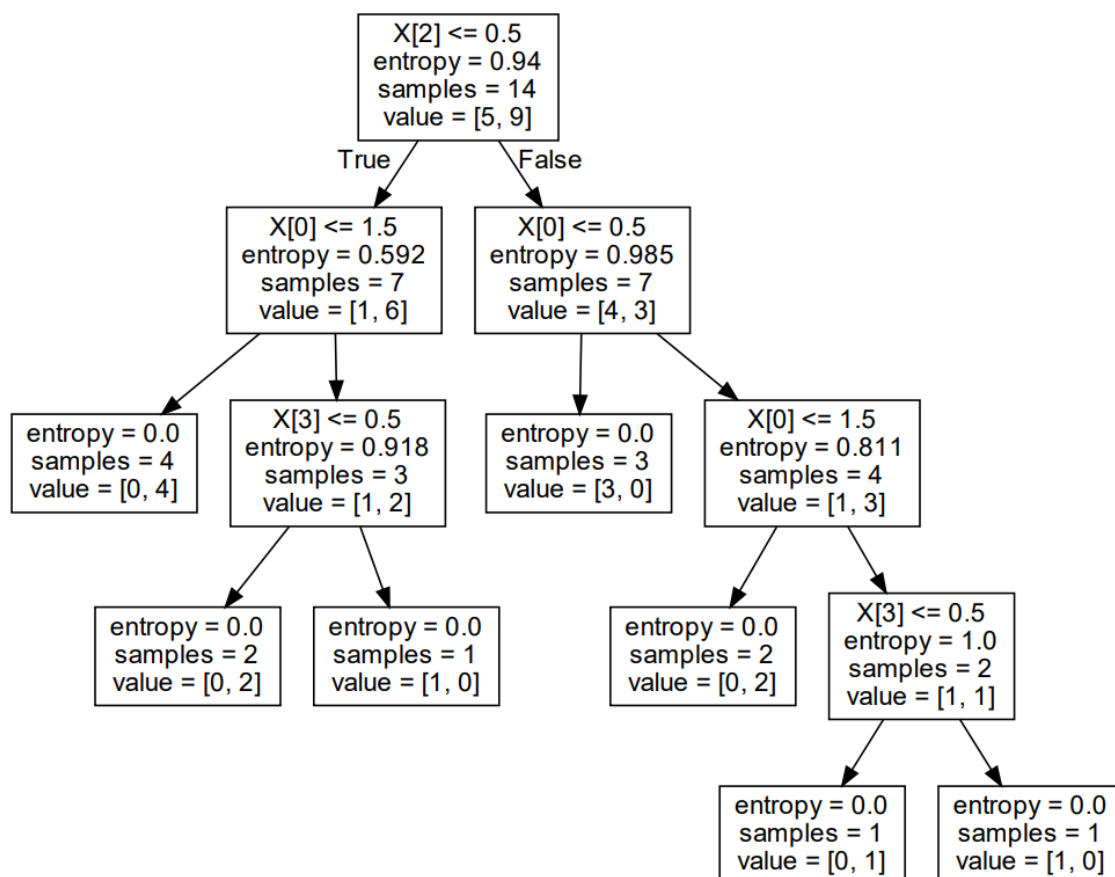
```
def data2vectoc(data):
    names = data.columns[:-1]
    for i in names:
        col = pd.Categorical(data[i])
        data[i] = col.codes
    return data
```

步骤 5 调用函数进行预测

输入：

```
data = pd.read_table("tennis.txt", header=None, sep='\t') #读取训练数据
trainingvec = data2vectoc(data) #向量化数据
decisionTree = createTree(trainingvec) #创建决策树
showtree2pdf(decisionTree, "tennis.pdf") #图示决策树
```

此时在本地生成决策树图，名称为“tennis.pdf”。



可以看到里面的内容就是决策树的可视化呈现。其中 $X[2]$ 就表示第 3 个特征变量：湿度， $X[0]$ 则表示第 1 个特征变量：天气， $X[3]$ 则表示第 4 个特征变量：风力；entropy 则表示该节点的熵值；samples 则表示该节点中的样本数，比如说第一个节点，也即根节点中的 14 就是训练集中的样本数量；value 则表示不同种类所占的个数，比如说根节点中 value 左边的 5 表示“否”的数量，9 则表示“是”的数量。

新样本预测，输入：

```
testVec = [0,0,1,1] # 天气晴、气温冷、湿度高、风力强
print(decisionTree.predict(np.array(testVec).reshape(1,-1))) #预测
```

输出：

```
['否']
```

1.2.5 朴素贝叶斯

通过 jieba 文字分词库对邮件数据集的垃圾邮件和进行文本处理，提取特征。然后调用 sklearn 机器学习库中的朴素贝叶斯算法训练模型，最后推理测试集中邮件是否为垃圾邮件。

步骤 1 引入相关依赖的包

输入：

```
from numpy import *
from os import listdir
```



```
import codecs #字符转换模块，用于文本的编码和解码
import jieba#中文分词库
import re
from sklearn.naive_bayes import MultinomialNB
from collections import Counter
from itertools import chain #用于串联迭代对象
```

步骤 2 构建文本处理函数

输入：

```
def segment2word(doc: str):
    #从 stop_list.txt 文件中提取停用词，存储为列表
    stop_words = codecs.open("./ML/04/stop_list.txt", "r", "UTF-8").read().splitlines()
    doc = re.sub('[\t\r\n]', ' ', doc)#去除邮件文本中的缩进，换行等
    word_list = list(jieba.cut(doc.strip())) #用 jieba 进行分词
    out_str = ""
    for word in word_list: #删去邮件文本中的停用词
        if word == ' ' or word == "":#
            continue
        if word not in stop_words:
            out_str += word.strip()
            out_str += ' '
    segments = out_str.strip().split(sep=' ')
    return segments
```

步骤 3 构建文本读取函数

输入：

```
def getDatafromDir(data_dir):
    docLists = []
    docLabels = [f for f in listdir(data_dir) if f.endswith('.txt') ]#存储每一封邮件的名称
    for doc in docLabels:
        try:
            filepath=data_dir + "/" + doc
            #对训练集的邮件进行文本处理
            wordList = segment2word(codecs.open(filepath, "r", "UTF-8").read())
            docLists.append(wordList)#整合训练集的邮件处理后的结果
        except:
            print("handling file %s is error!!" %filepath)
    return docLists
```

步骤 4 构建数据集

输入：

```
spamDocList=getDatafromDir("./ML/04/email/spam/") #对垃圾邮件进行文本处理
hamDocList = getDatafromDir("./ML/04/email/ham/") #对正常邮件进行文本处理
```

```

fullDocList = spamDocList + hamDocList#储存邮件的特征
# 添加标签，垃圾邮件标记为 1，正常邮件标记为 0
classList = array([1]*len(spamDocList)+[0]*len(hamDocList))
frequencyDic = Counter(chain(*fullDocList)) # 生成词频映射词典
topWords = [w[0] for w in frequencyDic.most_common(500)] #获取前 500 个最频繁的热词。
vector = []

for docList in fullDocList:
    #统计每封邮件中每个热词出现的频率
    topwords_list = list(map(lambda x: docList.count(x), topWords))
    vector.append(topwords_list)

#生成 vector 作为数据特征
vector = array(vector)

```

步骤 5 模型训练

输入：

```

model = MultinomialNB() #选取多项式贝叶斯为训练模型
model.fit(vector, classList) #vector 为特征，classList 为标签，训练贝叶斯模型

```

步骤 6 模型测试

输入：

```

#存储每一封训练集邮件的名称
dataList=[]
test_dir = "./ML/04/email/spam/"
docLabels = [f for f in listdir(test_dir) if f.endswith('.txt')]

#模型推理
for doc in docLabels:
    try:
        filepath = test_dir + "/" + doc
        dataList = segment2word(codecs.open(filepath, "r", "UTF-8").read())
    except:
        print("handling file %s is error!!" % filepath)

#统计测试集邮件中的热词的词频，提取特征
testVector = array(tuple(map(lambda x: dataList.count(x), topWords)))
testVector_reshape = testVector.reshape(1,-1)

#特征传入模型进行推理
predicted_label = model.predict(testVector.reshape(1, -1))
if(predicted_label == 1):
    print("%s is spam mail" %doc)
else:

```

```
print("%s is NOT spam mail" % doc)
```

输出：

```
ham134.txt is NOT spam mail
ham148.txt is NOT spam mail
ham22.txt is NOT spam mail
ham5.txt is NOT spam mail
spam017.txt is NOT spam mail
spam07.txt is spam mail
spam117.txt is spam mail
spam3.txt is spam mail
spam32.txt is spam mail
spam79.txt is spam mail
```

1.2.6 K-means 算法实现

步骤 1 导入依赖

make_blobs 用于产生本次实验所需的数据集，matplotlib 用于数据可视化，KMeans 用于 Kmeans 算法拟合训练。

输入：

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

步骤 2 生成数据集

make_blobs() 可以用来生成聚类任务的数据集，它会返回产生的数据集以及相应标签。但是需要注意的是，Kmean 算法是用于处理无标签数据集的，也就是，在实际使用 Kmeans 算法的时候，数据集是不含标签的。此处为方便大家理解，产生的数据集包含了标签信息。

其中，n_samples 表示样本数量，n_features 表示每个样本的特征数，也表示数据的维度，centers 表示类别数，random_state 表示随机生成器的种子，可以固定生成的数据，给定之后，每次生成的数据集就是固定的。

输入：

```
X, y = make_blobs(n_samples=500, n_features=2, centers=4, random_state=1)
```

查看产生的数据集的维度信息。

输入：

```
print("X 的维度为: {}".format(X.shape))
print("y 的维度为: {}".format(y.shape))
```

输出：

```
X 的维度为: (500, 2)
y 的维度为: (500,)
```

可以看到，该数据集一共包含 500 个样本点，每个样本点包含了 2 个特征。

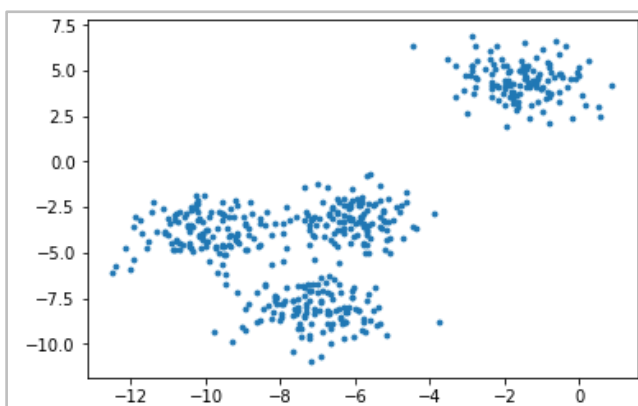
步骤 3 绘制散点图

首先，直接绘制该数据集的散点图，此时不考虑该数据集在生成时对应的标签。从图中我们可以看到 4 个不同的簇。

输入：

```
fig, ax1 = plt.subplots(1)
ax1.scatter(X[:, 0], X[:, 1]
            ,marker='o' # 设置点的形状为圆形
            ,s=8 # 设置点的大小
            )
plt.show()
```

输出：



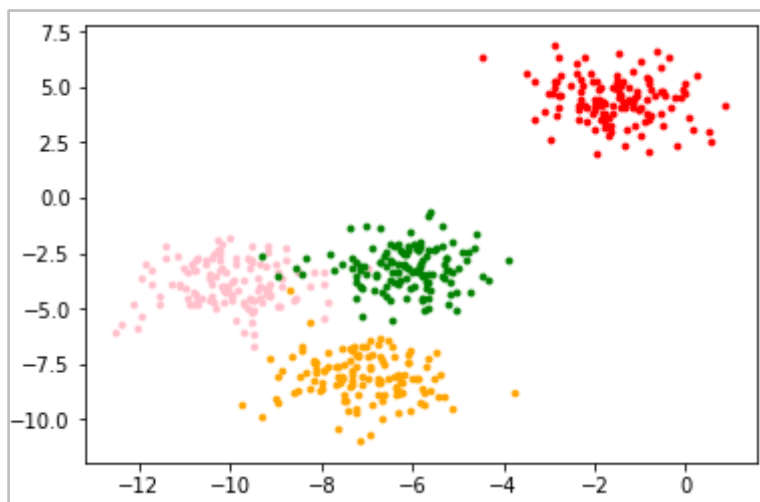
接下来，我们根据生成数据集时对应的标签，按照不同颜色绘制散点图。

输入：

```
color = ["red", "pink", "orange", "green"]
fig, ax1 = plt.subplots(1)

for i in range(4):
    ax1.scatter(X[y==i, 0], X[y==i, 1] # 根据每个点的标签绘制
                ,marker='o' # 设置点的形状为圆形
                ,s=8 # 设置点的大小
                ,c=color[i]
                )
plt.show()
```

输出：



步骤 4 Kmeans 聚类

调用 Sklearn 提供的 `sklearn.cluster.KMeans` 实现 K-means 聚类。首先我们将数据集聚成 3 类。

输入：

```
n_clusters = 3
cluster1 = KMeans(n_clusters=n_clusters,random_state=0).fit(X)
```

查看聚类后的每一个样本点的标签。

输入：

```
y_pred1 = cluster1.labels_
print(y_pred1)
```

输出：

```
[0 0 2 1 2 1 2 2 2 2 0 0 2 1 2 0 2 0 1 2 2 2 2 1 2 2 1 1 2 2 0 1 2 0 2 0 2
 2 0 2 2 2 1 2 2 0 2 2 1 1 1 2 2 2 0 2 2 2 2 2 1 1 2 2 1 2 0 2 2 2 0 2 2 0
 2 2 0 2 2 2 1 1 2 1 1 2 2 1 2 2 1 0 2 2 1 0 0 2 0 1 1 0 1 2 1 2 2 1 1 2 2
 0 1 2 1 2 1 2 1 2 2 0 0 2 2 2 1 0 0 2 1 2 2 2 2 0 1 2 1 1 2 0 2 1 1 1 2 2
 0 0 2 2 1 0 1 2 2 2 2 2 2 2 2 2 1 0 0 0 2 1 0 2 2 0 1 2 2 2 2 0 2 2 1 0 0
 2 2 0 0 2 1 1 0 0 2 1 2 0 0 1 0 2 1 2 2 0 2 2 0 2 2 2 2 2 0 2 2 1 2 1 2 0
 2 2 2 2 2 1 2 1 0 2 0 2 1 1 2 0 1 0 2 2 0 0 0 0 2 2 0 2 2 1 1 2 2 1 2 2 2
 1 2 1 2 2 1 2 0 0 2 2 2 2 1 1 2 1 2 0 1 0 1 0 0 1 0 1 1 2 2 2 2 2 2 0 1
 0 0 0 2 2 2 0 2 0 0 2 0 0 2 1 0 2 2 1 1 2 0 1 1 2 0 1 1 2 2 1 2 2 0 0 1 2
 0 2 1 1 2 2 2 0 2 1 1 2 1 1 1 1 0 0 2 1 2 2 0 1 2 1 2 1 2 2 2 1 2 2 0 1 0
 0 0 0 0 0 2 0 1 0 1 1 2 1 2 2 2 0 1 2 1 2 0 2 2 0 2 2 1 1 0 2 2 1 2 2 0 0
 2 0 2 2 0 2 0 2 1 0 1 2 2 1 2 2 1 0 2 1 1 2 2 2 2 0 1 0 2 1 0 0 0 2 1 2 0
 2 2 2 2 0 2 2 2 2 2 2 0 2 2 0 2 1 2 1 2 2 2 1 1 1 2 2 2 0 2 1 2 0 1 0 1 0
 2 1 1 0 2 2 0 2 2 2 0 2 1 2 2 0 0 0 2]
```

打印每个簇的质心。

输入：

```
centroid1 = cluster1.cluster_centers_
```

```
print(centroid1)
```

输出：

```
[[ -7.09306648 -8.10994454]
 [ -1.54234022  4.43517599]
 [ -8.0862351  -3.5179868 ]]
```

可视化聚类结果。

输入：

```
color = ["red","pink","orange","gray"]

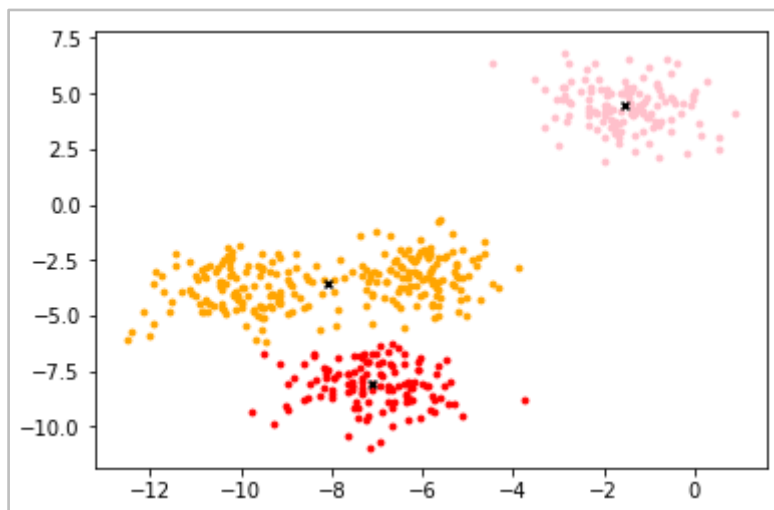
fig, ax1 = plt.subplots(1)

for i in range(n_clusters):
    ax1.scatter(X[y_pred1==i, 0], X[y_pred1==i, 1]
                ,marker='o' #点的形状
                ,s=8 #点的大小
                ,c=color[i]
                )

ax1.scatter(centroid1[:,0],centroid1[:,1]
            ,marker="x"
            ,s=15
            ,c="black")

plt.show()
```

输出：



可以看到，数据集被聚成了 3 个簇，每个簇的质心用黑色 x 表示。

步骤 5 再次 Kmeans 聚类

接下来，我们按照上述步骤重新实现 Kmeans 聚类，这次我们聚为 4 类。

输入：

```
n_clusters = 4
cluster2 = KMeans(n_clusters=n_clusters,random_state=0).fit(X)
y_pred2 = cluster2.labels_
centroid2 = cluster2.cluster_centers_
print("质心: {}".format(centroid2))
```

输出:

```
质心: [[ -6.08459039  -3.17305983]
 [ -1.54234022   4.43517599]
 [ -7.09306648  -8.10994454]
 [-10.00969056  -3.84944007]]
```

可视化聚类结果。

输入:

```
color = ["red","pink","orange","green"]

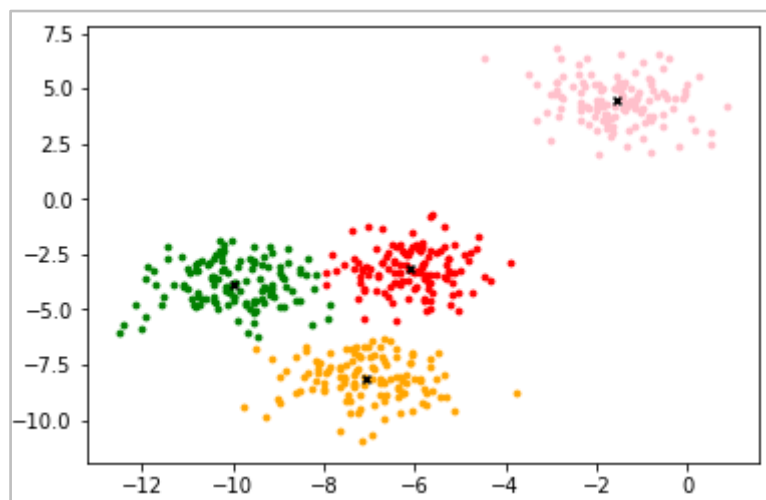
fig, ax1 = plt.subplots(1)

for i in range(n_clusters):
    ax1.scatter(X[y_pred2==i, 0], X[y_pred2==i, 1]
                ,marker='o' #点的形状
                ,s=8 #点的大小
                ,c=color[i]
                )

ax1.scatter(centroid2[:,0],centroid2[:,1]
            ,marker="x"
            ,s=15
            ,c="black")

plt.show()
```

输出:



可以看到，数据集被聚成了 4 个簇，每个簇的质心用黑色 x 表示。大家可以对比带有标签信息的原始数据散点图和被聚为 4 类后的数据散点图，可以发现，存在较多的样本点被聚到了错误的簇中。

1.3 思考题

如何使用 Python 从零实现 KNN 算法？

1.4 本章总结

本章主要介绍机器学习实现流程：主要包含导入数据，分割数据，数据标准化，定义模型以及设置相关超参数等方面；基于 sklearn 实现常用机器学习算法，加深学员整体上对机器学习模型的构建和使用的理解。

华为认证 AI 系列教程

HCIA-AI

深度学习和人工智能开发框架

实验指导手册

版本：3.5



华为技术有限公司

版权所有 © 华为技术有限公司 2022。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技术架构与应用认证、云服务与平台认证两类认证。

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。

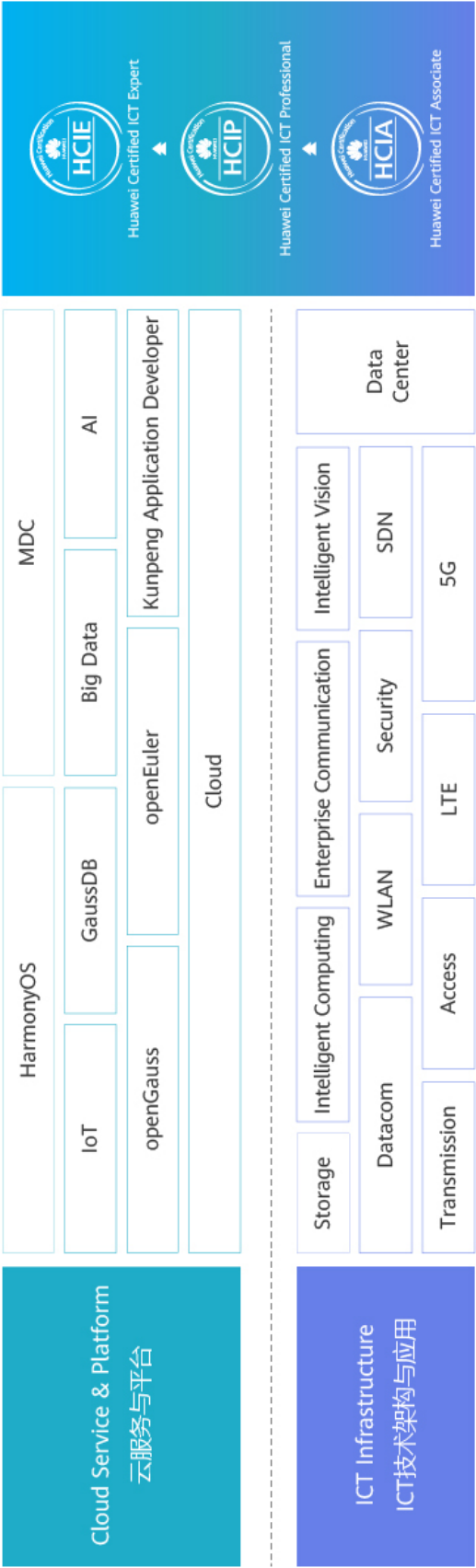
华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIA-AI V3.5定位于培养和认证具备使用机器学习、深度学习等算法进行AI产品和解决方案设计、开发和创新能力的工程师。

通过HCIA-AI V3.5认证，将证明您了解人工智能发展历史、华为昇腾AI体系和全栈全场景AI战略知识，掌握传统机器学习和深度学习的相关算法；具备利用MindSpore开发框架和MindSpore开发框架进行搭建、训练、部署神经网络的能力；能够胜任人工智能领域销售、市场、产品经理、项目管理、技术支持等岗位。

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！

华为认证



前言

简介

本书为 HCIA-AI 认证培训教程，适用于准备参加 HCIA-AI 考试的学员或者希望了解 AI 基础知识及 MindSpore 编程基础的读者。

内容描述

本实验指导书共包含 5 个实验：

- 实验一为 MindSpore 基础，本实验会主要介绍 MindSpore 的基本语法和常用模块。
- 实验二为手写字体图像识别实验，使用 MindSpore 框架实现手写字体的识别。
- 实验三为 MobileNet V2 图像分类实验，主要介绍使用轻量级网络 MobileNet V2 实现花卉图像进行分类。
- 实验四为 ResNet50 图像分类实验，主要介绍利用 ResNet50 模型对花卉图像进行分类。
- 实验五为 TextCNN 情感分析实验，主要介绍利用 TextCNN 模型对语句进行情感分析。

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 知识背景，同时熟悉 MindSpore 基本概念，了解基本 Python 编程知识。
-

目录

前 言	3
简介	3
内容描述	3
读者知识背景	3
1 MindSpore 基础	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 实验环境	6
1.2 实验步骤	6
1.2.1 Tensor 介绍	6
1.2.2 数据集加载	11
1.2.3 网络构建	14
1.2.4 模型训练与评估	17
1.2.5 模型保存与加载	18
1.2.6 自动微分	19
1.3 思考题	21
2 MNIST 手写体字符实验	22
2.1 实验介绍	22
2.1.1 关于本实验	22
2.2 实验准备	22
2.3 实验详细设计与实现	22
2.3.1 数据准备	22
2.3.2 实验步骤	23
3 MobileNet V2 图像分类实验	29
3.1 实验介绍	29
3.2 实验准备	29
3.3 实验详细设计与实现	29
3.3.1 数据准备	29
3.3.2 实验步骤	30

3.4 思考题	40
4 ResNet50 图像分类实验	41
4.1 实验介绍	41
4.2 实验准备	41
4.3 实验详细设计与实现	41
4.3.1 数据准备	41
4.3.2 实验步骤	42
4.4 思考题	56
5 TextCNN 情感分析实验	57
5.1 实验介绍	57
5.2 实验准备	57
5.3 实验详细设计与实现	57
5.3.1 数据准备	57
5.3.2 实验步骤	58
5.4 思考题	68

1 MindSpore 基础

1.1 实验介绍

1.1.1 关于本实验

本实验主要是介绍 MindSpore 的张量数据结构，通过对张量的一系列操作介绍，可以使学员对 MindSpore 的基本语法有所了解。

1.1.2 实验目的

- 掌握张量的创建方法。
- 掌握张量的属性与方法。

1.1.3 实验环境

本实验所需的框架为 MindSpore，建议版本为 1.7 及以上版本。实验可以在个人 PC 上完成也可以登录华为云购买 ModelArts 服务完成。

1.2 实验步骤

1.2.1 Tensor 介绍

张量（Tensor）是 MindSpore 网络运算中的基本数据结构。张量中的数据类型可参考 [dtype](#)。

不同维度的张量分别表示不同的数据，0 维张量表示标量，1 维张量表示向量，2 维张量表示矩阵，3 维张量可以表示彩色图像的 RGB 三通道等等。

MindSpore 张量支持不同的数据类型，包含 int8、int16、int32、int64、uint8、uint16、uint32、uint64、float16、float32、float64、bool_，与 NumPy 的数据类型一一对应。

在 MindSpore 的运算处理流程中，Python 中的 int 数会被转换为定义的 int64 类型，float 数会被转换为定义的 float32 类型。

1.2.1.1 创建 tensor

构造张量时，支持传入 Tensor、float、int、bool、tuple、list 和 NumPy.array 类型，其中 tuple 和 list 里只能存放 float、int、bool 类型数据。

Tensor 初始化时，可指定 dtype。如果没有指定 dtype，初始值 int、float、bool 分别生成数据类型为 mindspore.int32、mindspore.float32、mindspore.bool_ 的 0 维 Tensor，初始值 tuple 和 list 生成的 1 维 Tensor 数据类型与 tuple 和 list 里存放的数据类型相对应，如果包含多种不同类型的数据，则按照优先级：bool < int < float，选择相对优先级最高类型所对应的 mindspore 数据类型。如果初始值是 Tensor，则生成的 Tensor 数据类型与其一致；如果初始值是 NumPy.array，则生成的 Tensor 数据类型与之对应。

步骤 1 数组创建 Tensor

代码：

```
# 导入 MindSpore
import mindspore
# cell 同时输出多行
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import numpy as np
from mindspore import Tensor
from mindspore import dtype
# 用数组创建张量
x = Tensor(np.array([[1, 2], [3, 4]]), dtype.int32)
x
```

输出：

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

步骤 2 用数值创建 Tensor

代码：

```
# 用数值创建张量
y = Tensor(1.0, dtype.int32)
z = Tensor(2, dtype.int32)
y
z
```

输出：

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

步骤 3 用 Bool 创建 Tensor

代码：

```
# 用 Bool 创建张量
m = Tensor(True, dtype.bool_)
m
```

输出：

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

步骤 4 用 tuple 创建 Tensor

代码：

```
# 用 tuple 创建张量
n = Tensor((1, 2, 3), dtype.int16)
n
```

输出

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

步骤 5 用 list 创建 Tensor

代码：

```
# 用 list 创建张量
p = Tensor([4.0, 5.0, 6.0], dtype.float64)
p
```

输出：

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

步骤 6 继承另一个张量的属性，形成新的张量

代码：

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

输出：

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

步骤 7 输出恒定值 Tensor

代码：

```
from mindspore.ops import operations as ops

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

zeros = ops.Zeros()
```

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

输出：

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

1.2.1.2 Tensor 的属性

张量的属性包括形状（shape）和数据类型（dtype）。

- 形状：Tensor 的 shape，是一个 tuple。
- 数据类型：Tensor 的 dtype，是 MindSpore 的一个数据类型。

代码：

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

```
x.shape # 形状
x.dtype # 数据类型
x.ndim  # 维度
x.size  # 大小
```

输出：

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

1.2.1.3 Tensor 的方法

asnumpy()：将 Tensor 转换为 NumPy 的 array。

代码：

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

```
# 将 Tensor 数据类型转换成 NumPy
y_array = y.asnumpy()

y
y_array
```

输出：

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

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

1.2.1.4 Tensor 的运算

张量之间有很多运算，包括算术、线性代数、矩阵处理（转置、标引、切片）、采样等，下面介绍其中几种操作，张量运算和 NumPy 的使用方式类似。

步骤 1 索引和切片

代码：

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

输出：

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

步骤 2 Tensor 拼接

代码：

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

输出：

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

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

步骤 3 转换为 Numpy

代码：

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

输出：

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

1.2.2 数据集加载

MindSpore.dataset 提供 API 来加载和处理各种常见的数据集，如 MNIST, CIFAR-10, CIFAR-100, VOC, ImageNet, CelebA 等。

步骤 1 加载 MNIST 数据集

加载数据集之前，建议先从以下链接 <http://yann.lecun.com/exdb/mnist/>，下载 MNIST 数据集，并将训练文件和测试文件放到 MNIST 文件夹下面。

代码：

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

dataset_dir = "./MNIST/train" # 数据集路径
# 从 mnist dataset 读取 3 张图片
mnist_dataset = ds.MnistDataset(dataset_dir=dataset_dir, num_samples=3)
# 查看图像，设置图像大小
plt.figure(figsize=(8,8))
i = 1

# 打印 3 张子图
for dic in mnist_dataset.create_dict_iterator(output_numpy=True):
    plt.subplot(3,3,i)
    plt.imshow(dic['image'][:, :, 0])
    plt.axis('off')
    i += 1
plt.show()
```

输出：



图1-1 MNIST 数据集样本

步骤 2 自定义数据集

对于目前 MindSpore 不支持直接加载的数据集，可以构造自定义数据集类，然后通过 GeneratorDataset 接口实现自定义方式的数据加载。

代码：

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

```
class DatasetGenerator:
#实例化数据集对象时，__init__函数被调用，用户可以在此进行数据初始化等操作。
    def __init__(self):
        self.data = np.random.sample((5, 2))
        self.label = np.random.sample((5, 1))
#定义数据集类的__getitem__函数，使其支持随机访问，能够根据给定的索引值 index，获取数据集中的数据并返回。
    def __getitem__(self, index):
        return self.data[index], self.label[index]
#定义数据集类的__len__函数，返回数据集的样本数量。
    def __len__(self):
        return len(self.data)
#定义数据集类之后，就可以通过 GeneratorDataset 接口按照用户定义的方式加载并访问数据集样本。
dataset_generator = DatasetGenerator()
dataset = ds.GeneratorDataset(dataset_generator, ["data", "label"], shuffle=False)
#通过 create_dict_iterator 方法获取数据
for data in dataset.create_dict_iterator():
    print('{0}'.format(data["data"]), '{0}'.format(data["label"]))
```

输出：

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

步骤 3 数据增强

MindSpore 提供的数据集接口具备常用的数据处理方法，如：打乱、设置 batch 等。用户只需调用相应的函数接口即可快速进行数据处理。

下面的样例先将数据集随机打乱顺序，然后将样本两两组成一个批次。

代码：

```
ds.config.set_seed(58)

# 随机打乱数据顺序，buffer_size 表示数据集中进行 shuffle 操作的缓存区的大小。
dataset = dataset.shuffle(buffer_size=10)
# 对数据集进行分批，batch_size 表示每组包含的数据个数，现设置每组包含 2 个数据。
dataset = dataset.batch(batch_size=2)

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

输出：

```
data: [[0.36510558 0.45120592]
```

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

代码：

```
import matplotlib.pyplot as plt

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

DATA_DIR = './MNIST/train'
#取出 6 个样本
mnist_dataset = ds.MnistDataset(DATA_DIR, num_samples=6, shuffle=False)
# 查看数据原图
mnist_it = mnist_dataset.create_dict_iterator()
data = next(mnist_it)
plt.imshow(data['image'].asnumpy().squeeze(), cmap=plt.cm.gray)
plt.title(data['label'].asnumpy(), fontsize=20)
plt.show()
```

输出：

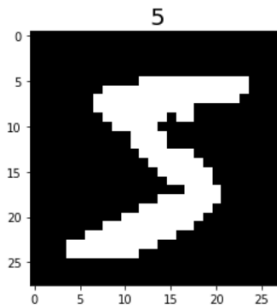


图1-2 数据样本

代码：

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

输出：

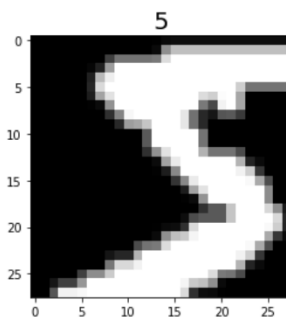


图1-3 数据增强后的效果

1.2.3 网络构建

MindSpore 将构建网络层的接口封装在 nn 模块中，我们将通过调用来构建不同类型的神经网络层。

步骤 1 全连接层

全连接层：mindspore.nn.Dense

- in_channels: 输入通道
- out_channels: 输出通道
- weight_init: 权重初始化, Default 'normal'.

代码：

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

# 构造输入张量
input_a = Tensor(np.array([[1, 1, 1], [2, 2, 2]]), ms.float32)
print(input_a)
# 构造全连接网络，输入通道为 3，输出通道为 3
net = nn.Dense(in_channels=3, out_channels=3, weight_init=1)
output = net(input_a)
print(output)
```

输出：

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

步骤 2 卷积层

代码：


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

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

输出：

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

步骤 3 ReLU 层

代码：

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

print(output)
```

输出：

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

步骤 4 池化层

代码：

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

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

输出：

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

步骤 5 Flatten 层

代码：

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

print(output.shape)
```

输出：

```
(1, 400)
```

步骤 6 定义模型类并查看参数

MindSpore 的 Cell 类是构建所有网络的基类，也是网络的基本单元。当用户需要神经网络时，需要继承 Cell 类，并重写 `__init__` 方法和 `construct` 方法。

代码：

```
class LeNet5(nn.Cell):
    """
    Lenet 网络结构
    """
    def __init__(self, num_class=10, num_channel=1):
        super(LeNet5, self).__init__()
        # 定义所需要的运算
        self.conv1 = nn.Conv2d(num_channel, 6, 5, pad_mode='valid')
        self.conv2 = nn.Conv2d(6, 16, 5, pad_mode='valid')
        self.fc1 = nn.Dense(16 * 4 * 4, 120)
        self.fc2 = nn.Dense(120, 84)
        self.fc3 = nn.Dense(84, num_class)
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()

    def construct(self, x):
        # 使用定义好的运算构建前向网络
        x = self.conv1(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x

#实例化模型，利用 parameters_and_names 方法查看模型的参数
modelle = LeNet5()
for m in modelle.parameters_and_names():
    print(m)
```

输出：

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

1.2.4 模型训练与评估

步骤 1 损失函数

损失函数用来评价模型的预测值和真实值不一样的程度，在这里，使用绝对误差损失函数 L1Loss。mindspore.nn.loss 也提供了许多其他常用的损失函数，如 SoftmaxCrossEntropyWithLogits、MSELoss、SmoothL1Loss 等。

我们给定输出值和目标值，计算损失值，使用方法如下所示：

代码：

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

输出：

```
1.5
```

步骤 2 优化器

深度学习优化算法大概常用的有 SGD、Adam、Ftrl、lazyadam、Momentum、RMSprop、Lars、Proximal_ada_grad 和 lamb 这几种。

动量优化器：mindspore.nn.Momentum

代码

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

步骤 3 模型编译

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

代码：

```
from mindspore import Model

# 定义神经网络
net = LeNet5()
# 定义损失函数
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
# 定义优化器
optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
# 模型编译
```

```
model = Model(network = net, loss_fn=loss, optimizer=optim, metrics={'accuracy'})
```

步骤 4 模型训练

代码：

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

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

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

mnist_dataset = mnist_dataset.map(input_columns="image", operations=[rescale_op,resize_op,
hwc2chw_op])
mnist_dataset = mnist_dataset.map(input_columns="label", operations=C.TypeCast(ms.int32))
mnist_dataset = mnist_dataset.batch(32)
loss_cb = LossMonitor(per_print_times=1000)
# dataset 是传入参数代表训练集，epoch 是训练集迭代轮数
model.train(epoch=1, train_dataset=mnist_dataset,callbacks=[loss_cb])
```

步骤 5 模型评估

代码：

```
# 测试集
DATA_DIR = './forward_mnist/MNIST/test'
dataset = ds.MnistDataset(DATA_DIR)

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

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

1.2.5 模型保存与加载

在上述网络模型训练完成后，可以通过两种形式对模型进行保存：

1. 简单的对网络模型进行保存，可以在训练前后进行保存。这种方式的优点是接口简单易用，但是只保留执行命令时候的网络模型状态；

代码：

```
import mindspore as ms
```

```
# 定义的网络模型为 net，一般在训练前或者训练后使用
ms.save_checkpoint(net, "./MyNet.ckpt") # net 为训练网络，"./MyNet.ckpt"为网络模型的保存路径。
```

输出：

```
epoch: 1 step: 1000, loss is 2.283555746078491
```

2. 在网络模型训练中进行保存，MindSpore 在网络模型训练的过程中，自动保存训练时候设定好的 epoch 数和 step 数的参数，也就是把模型训练过程中产生的中间权重参数也保存下来，方便进行网络微调和停止训练。

代码：

```
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig

# 设置 epoch_num 数量
epoch_num = 5

# 设置模型保存参数
config_ck = CheckpointConfig(save_checkpoint_steps=1875, keep_checkpoint_max=10)

# 应用模型保存参数
ckptpoint = ModelCheckpoint(prefix="lenet", directory="./lenet", config=config_ck)
model.train(epoch_num, mnist_dataset, callbacks=[ckptpoint])
```

1.2.6 自动微分

在训练神经网络时，最常用的算法是反向传播，在该算法中，根据损失函数对于给定参数的梯度来调整参数（模型权重）。

MindSpore 计算一阶导数方法 `mindspore.ops.GradOperation (get_all=False, get_by_list=False, sens_param=False)`，其中 `get_all` 为 `False` 时，只会对第一个输入求导，为 `True` 时，会对所有输入求导；`get_by_list` 为 `False` 时，不会对权重求导，为 `True` 时，会对权重求导；`sens_param` 对网络的输出值做缩放以改变最终梯度。

下面用 `MatMul` 算子的求导做深入分析。

步骤 1 对输入求一阶导数

如果需要对输入进行求导，首先需要定义一个需要求导的网络，以一个由 `MatMul` 算子构成的网络 $f(x,y)=z*x*y$ 为例。

代码：

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

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

    def construct(self, x, y):
        x = x * self.z
        out = self.matmul(x, y)
        return out

class GradNetWrtX(nn.Cell):
    def __init__(self, net):
        super(GradNetWrtX, self).__init__()
        self.net = net
        self.grad_op = ops.GradientOperation()

    def construct(self, x, y):
        gradient_function = self.grad_op(self.net)
        return gradient_function(x, y)

x = Tensor([[0.8, 0.6, 0.2], [1.8, 1.3, 1.1]], dtype=mstype.float32)
y = Tensor([[0.11, 3.3, 1.1], [1.1, 0.2, 1.4], [1.1, 2.2, 0.3]], dtype=mstype.float32)
output = GradNetWrtX(Net())(x, y)
print(output)
```

输出：

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

步骤 2 对权重求一阶导数

若需要对权重的求导，将 `ops.GradientOperation` 中的 `get_by_list` 设置为 `True`。若需要对某些权重不进行求导，则在定义求导网络时，对相应的权重中 `requires_grad` 设置为 `False`。

代码：

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

    def construct(self, x, y):
        gradient_function = self.grad_op(self.net, self.params)
        return gradient_function(x, y)

output = GradNetWrtX(Net())(x, y)
print(output)
```

输出：

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

1.3 思考题

创建两个 Tensor，t1 和 t2，通过以下代码 t1 能否正常创建？比较两个 Tensor 的输出是否一样，如果不一样有什么区别？

```
t1 = Tensor(np.array([[1.2, 2.2], [3.2, 4.2]]), dtype.int32)  
t2 = Tensor(np.array([[1.2, 2.2], [3.2, 4.2]]), dtype.float32)
```

2 MNIST 手写体字符实验

2.1 实验介绍

2.1.1 关于本实验

本实验实现的是深度学习领域的经典案例——MNIST 手写体字符识别实验。整体流程如下：

- 处理需要的数据集，这里使用了 MNIST 数据集。
- 定义一个网络，这里自己搭建了简单的全连接网络。
- 定义损失函数和优化器。
- 加载数据集并进行训练，训练完成后，利用测试集进行评估。

2.2 实验准备

在动手进行实践之前，确保你已经正确安装了 MindSpore。如果没有，可以通过 MindSpore 官网安装页面：<https://www.mindspore.cn/install/>，将 MindSpore 安装在你的电脑当中。

同时希望你拥有 Python 编程基础和概率、矩阵等基础数学知识。

推荐环境：

版本：MindSpore 1.7

编程语言：Python 3.7

2.3 实验详细设计与实现

2.3.1 数据准备

我们示例中用到的 MNIST 数据集是由 10 类 28*28 的灰度图片组成，训练数据集包含 60000 张图片，测试数据集包含 10000 张图片。

MNIST 数据集下载页面：<http://yann.lecun.com/exdb/mnist/>。页面提供 4 个数据集下载链接，其中前 2 个文件是测试数据，后 2 个文件是训练数据。

将数据集下载并解压到本地路径下，这里将数据集解压分别存放到工作区的./MNIST /train、./MNIST /test 路径下。

目录结构如下：

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

2.3.2 实验步骤

步骤 1 导入 Python 库&模块并配置运行信息

在使用前，导入需要的 Python 库。

目前使用到 os 库，为方便理解，其他需要的库，我们在具体使用到时再说明。详细的 MindSpore 的模块说明，可以在 MindSpore API 页面中搜索查询。可以通过 [context.set_context](#) 来配置运行需要的信息，譬如运行模式、后端信息、硬件等信息。

导入 context 模块，配置运行需要的信息。

代码：

```
#导入相关依赖库
import os
from matplotlib import pyplot as plt
import numpy as np

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

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

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

本实验中我们的运行模式是图模式。根据实际情况配置硬件信息，譬如代码运行在 Ascend AI 处理器上，则 device_target 选择 Ascend，代码运行在 CPU、GPU 同理。详细参数说明，请参见 context.set_context 接口说明。

步骤 2 数据读取

利用 MindSpore 的数据读取功能读取 MNIST 数据集，并查看训练集和测试集数据量、样本信息。

代码：

```
DATA_DIR_TRAIN = "MNIST/train" # 训练集信息
DATA_DIR_TEST = "MNIST/test" # 测试集信息
#读取数据
ds_train = ds.MnistDataset(DATA_DIR_TRAIN)
ds_test = ds.MnistDataset(DATA_DIR_TEST)
#显示数据集的相关特性
print('训练数据集数量: ',ds_train.get_dataset_size())
print('测试数据集数量: ',ds_test.get_dataset_size())
image=ds_train.create_dict_iterator().__next__()
print('图像长/宽/通道数: ',image['image'].shape)
print('一张图像的标签样式: ',image['label']) #一共 10 类，用 0-9 的数字表达类别。
```

步骤 3 数据处理

数据集对于训练非常重要，好的数据集可以有效提高训练精度和效率。在加载数据集前，我们通常会对数据集进行一些处理。

定义数据集及数据操作

我们定义一个函数 create_dataset 来创建数据集。在这个函数中，我们定义好需要进行的数据增强和处理操作：

- 读取数据集。
- 定义进行数据增强和处理所需要的一些参数。
- 根据参数，生成对应的数据增强操作。
- 使用 map 映射函数，将数据操作应用到数据集。
- 对生成的数据集进行处理。

代码：

```
def create_dataset(training=True, batch_size=128, resize=(28, 28),
                  rescale=1/255, shift=0, buffer_size=64):
    ds = ms.dataset.MnistDataset(DATA_DIR_TRAIN if training else DATA_DIR_TEST)
    # 定义 Map 操作尺寸缩放，归一化和通道变换
    resize_op = CV.Resize(resize)
    rescale_op = CV.Rescale(rescale,shift)
    hwc2chw_op = CV.HWC2CHW()
    # 对数据集进行 map 操作
    ds = ds.map(input_columns="image", operations=[rescale_op,resize_op, hwc2chw_op])
    ds = ds.map(input_columns="label", operations=C.TypeCast(ms.int32))
    #设定打乱操作参数和 batchsize 大小
    ds = ds.shuffle(buffer_size=buffer_size)
```

```
ds = ds.batch(batch_size, drop_remainder=True)
return ds
```

其中，batch_size：每组包含的数据个数，现设置每组包含 32 个数据。先进行修改图片尺寸，归一化，修改图像通道等工作，再修改标签的数据类型。最后进行 shuffle 操作，同时设定 batch_size，设置 drop_remainder 为 True，则数据集中不足最后一个 batch 的数据会被抛弃。

MindSpore 支持进行多种数据处理和增强的操作，各种操作往往组合使用，具体可以参考数据处理与数据增强章节。

步骤 4 样本可视化

读取前 10 个样本，然后进行样本可视化，以此来确定样本是不是真实数据集。

代码：

```
#显示前 10 张图片以及对应标签,检查图片是否是正确的数据集
ds = create_dataset(training=False)
data = ds.create_dict_iterator().__next__()
images = data['image'].asnumpy()
labels = data['label'].asnumpy()
plt.figure(figsize=(15,5))
for i in range(1,11):
    plt.subplot(2, 5, i)
    plt.imshow(np.squeeze(images[i]))
    plt.title('Number: %s' % labels[i])
    plt.xticks([])
plt.show()
```

输出：

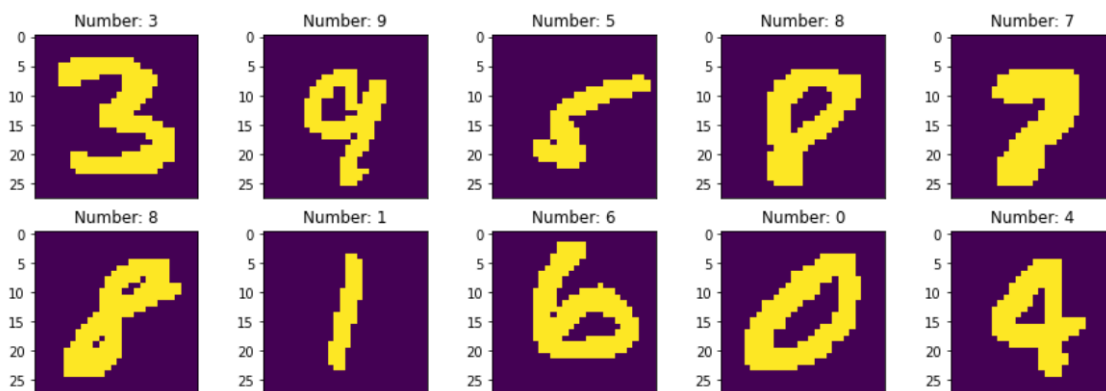


图2-1 样本可视化

步骤 5 定义网络

我们通过定义一个简单的全连接网络来完成图像识别，网络只有 3 层：

第一层全连接层，形状为 784*512；

第二层全连接层，形状为 512*128；

最后一层输出层，形状为 128*10。

使用 MindSpore 定义神经网络需要继承 mindspore.nn.Cell。Cell 是所有神经网络（Conv2d 等）的基类。

神经网络的各层需要预先在__init__方法中定义，然后通过定义 construct 方法来完成神经网络的前向构造。定义网络各层如下：

代码：

```
#创建模型。模型包括 3 个全连接层，最后输出层使用 softmax 进行多分类，共分成（0-9）10 类
class ForwardNN(nn.Cell):
    def __init__(self):
        super(ForwardNN, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Dense(784, 512, activation='relu')
        self.fc2 = nn.Dense(512, 128, activation='relu')
        self.fc3 = nn.Dense(128, 10, activation=None)

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

步骤 6 定义损失函数及优化器

损失函数：又叫目标函数，用于衡量预测值与实际值差异的程度。深度学习通过不停地迭代来缩小损失值。定义一个好的损失函数，可以有效提高模型的性能。

优化器：用于最小化损失函数，从而在训练过程中改进模型。

定义了损失函数后，可以得到损失函数关于权重的梯度。梯度用于指示优化器优化权重的方向，以提高模型性能。MindSpore 支持的损失函数有 SoftmaxCrossEntropyWithLogits、L1Loss、MSELoss 等。这里使用 SoftmaxCrossEntropyWithLogits 损失函数。

MindSpore 提供了 callback 机制，可以在训练过程中执行自定义逻辑，这里使用框架提供的 ModelCheckpoint 为例。ModelCheckpoint 可以保存网络模型和参数，以便进行后续的 fine-tuning（微调）操作。

代码：

```
#创建网络，损失函数，评估指标 优化器，设定相关超参数
lr = 0.001
num_epoch = 10
momentum = 0.9

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

```
opt = nn.Adam(net.trainable_params(), lr)
```

步骤 7 开始训练

训练过程指的是将训练集数据传入网络对网络中的参数进行训练优化的过程。在 MindSpore 框架中 Model.train 方法用于完成这一过程。

代码：

```
#编译模型
model = Model(net, loss, opt, metrics)
config_ck = CheckpointConfig(save_checkpoint_steps=1875, keep_checkpoint_max=10)
ckptpoint_cb = ModelCheckpoint(prefix="checkpoint_net",directory = "./ckpt" ,config=config_ck)
#生成数据集
ds_eval = create_dataset(False, batch_size=32)
ds_train = create_dataset(batch_size=32)
#训练模型
loss_cb = LossMonitor(per_print_times=1875)
time_cb = TimeMonitor(data_size=ds_train.get_dataset_size())
print("===== Starting Training =====")
model.train(num_epoch, ds_train,callbacks=[ckptpoint_cb,loss_cb,time_cb ],dataset_sink_mode=False)
```

训练过程中会打印 loss 值，类似下图。loss 值会波动，但总体来说 loss 值会逐步减小，精度逐步提高。每个人运行的 loss 值有一定随机性，不一定完全相同。训练过程中 loss 打印示例如下：

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

步骤 8 模型评估

本步骤利用原测试集去进行模型的评估。

代码：

```
#使用测试集评估模型，打印总体准确率  
metrics=model.eval(ds_eval)  
print(metrics)
```

输出：

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

3 MobileNet V2 图像分类实验

3.1 实验介绍

本实验将使用轻量级网络 MobileNet V2 实现花卉图像数据集分类。

3.2 实验准备

在动手进行实践之前，确保你已经正确安装了 MindSpore。如果没有，可以通过 MindSpore 官网安装页面：<https://www.mindspore.cn/install/>，将 MindSpore 安装在你的电脑当中。

同时希望你拥有 Python 编程基础和概率、矩阵等基础数学知识。

推荐环境：

版本：MindSpore 1.7

编程语言：Python 3.7

3.3 实验详细设计与实现

3.3.1 数据准备

我们示例中用到的图像花卉数据集，该数据集是开源数据集，总共包括 5 种花的类型：分别是 daisy（雏菊，633 张），dandelion（蒲公英，898 张），roses（玫瑰，641 张），sunflowers（向日葵，699 张），tulips（郁金香，799 张），保存在 5 个文件夹当中，总共 3670 张，大小大概在 230M 左右。为了在模型部署上线之后进行测试，数据集在这里分成了 flower_photos_train 和 flower_photos_test 两部分。

目录结构如下：

```
flower_photos_train
├── daisy
├── dandelion
├── roses
├── sunflowers
```

```

|—— tulips
|—— LICENSE.txt
flower_photos_test
|—— daisy
|—— dandelion
|—— roses
|—— sunflowers
|—— tulips
|—— LICENSE.txt

```

数据集的获取：

```

https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower\_photos\_train.zip
https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower\_photos\_test.zip

```

3.3.2 实验步骤

步骤 1 加载数据集

定义函数`create_dataset`加载数据集，使用`ImageFolderDataset`接口来加载花卉图像分类数据集，并对数据集进行图像增强操作。代码：

```

import mindspore.dataset as ds
import mindspore.dataset.vision.c_transforms as CV
from mindspore import dtype as mstype

train_data_path = 'flower_photos_train'
val_data_path = 'flower_photos_test'

def create_dataset(data_path, batch_size=18, training=True):
    """定义数据集"""

    data_set = ds.ImageFolderDataset(data_path, num_parallel_workers=8, shuffle=True,
                                     class_indexing={'daisy': 0, 'dandelion': 1, 'roses': 2, 'sunflowers':
3,
                                     'tulips': 4})

    # 对数据进行增强操作
    image_size = 224
    mean = [0.485 * 255, 0.456 * 255, 0.406 * 255]
    std = [0.229 * 255, 0.224 * 255, 0.225 * 255]
    if training:
        trans = [
            CV.RandomCropDecodeResize(image_size, scale=(0.08, 1.0), ratio=(0.75, 1.333)),

```



```

        CV.RandomHorizontalFlip(prob=0.5),
        CV.Normalize(mean=mean, std=std),
        CV.HWC2CHW()
    ]
else:
    trans = [
        CV.Decode(),
        CV.Resize(256),
        CV.CenterCrop(image_size),
        CV.HWC2CHW()
    ]

# 实现数据的 map 映射、批量处理和数据重复的操作
data_set = data_set.map(operations=trans, input_columns="image", num_parallel_workers=8)
# 设置 batch_size 的大小，若最后一次抓取的样本数小于 batch_size，则丢弃
data_set = data_set.batch(batch_size, drop_remainder=True)

return data_set

dataset_train = create_dataset(train_data_path)
dataset_val = create_dataset(val_data_path)

```

步骤 2 数据集可视化

从`create_dataset`接口中加载的训练数据集返回值为字典，用户可通过`create_dict_iterator`接口创建数据迭代器，使用`next`迭代访问数据集。本章中`batch_size`设为 18，所以使用`next`一次可获取 18 个图像及标签数据。代码：

```

import matplotlib.pyplot as plt
import numpy as np

data = next(dataset_train.create_dict_iterator())
images = data["image"]
labels = data["label"]

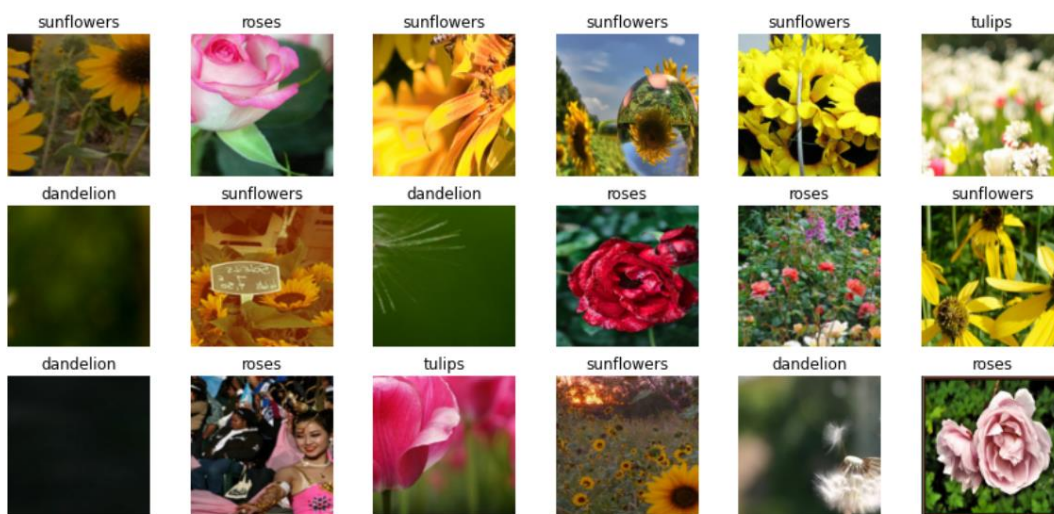
print("Tensor of image", images.shape)
print("Labels:", labels)

# class_name 对应 label，按文件夹字符串从小到大的顺序标记 label
class_name = {0:'daisy',1:'dandelion',2:'roses',3:'sunflowers',4:'tulips'}
plt.figure(figsize=(15, 7))
for i in range(len(labels)):
    # 获取图像及其对应的 label
    data_image = images[i].asnumpy()
    data_label = labels[i]
    # 处理图像供展示使用

```

```
data_image = np.transpose(data_image, (1, 2, 0))
mean = np.array([0.485, 0.456, 0.406])
std = np.array([0.229, 0.224, 0.225])
data_image = std * data_image + mean
data_image = np.clip(data_image, 0, 1)
# 显示图像
plt.subplot(3, 6, i + 1)
plt.imshow(data_image)
plt.title(class_name[int(labels[i].asnumpy())])
plt.axis("off")
plt.show()
```

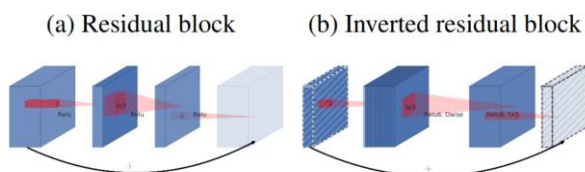
输出结果：



步骤 3 创建 MobileNet V2 模型

数据集对于训练非常重要，好的数据集可以有效提高训练精度和效率。在 MobileNet 网络是由 Google 团队于 2017 年提出的专注于移动端、嵌入式或 IoT 设备的轻量级 CNN 网络，相比于传统的卷积神经网络，MobileNet 网络使用深度可分离卷积（Depthwise Separable Convolution）的思想在准确率小幅度降低的前提下，大大减小了模型参数与运算量。并引入宽度系数 α 和分辨率系数 β 使模型满足不同应用场景的需求。

由于 MobileNet 网络中 Relu 激活函数处理低维特征信息时会存在大量的丢失，所以 MobileNetV2 网络提出使用倒残差结构（Inverted residual block）和 Linear Bottlenecks 来设计网络，以提高模型的准确率，且优化后的模型更小。



上图中 Inverted residual block 结构是先使用 1x1 卷积进行升维，然后使用 3x3 的 DepthWise 卷积，最后使用 1x1 的卷积进行降维，与 Residual block 结构相反。Residual block 是先使用 1x1 的卷积进行降维，然后使用 3x3 的卷积，最后使用 1x1 的卷积进行升维。

详细内容可参见 MobileNet V2 论文(<https://arxiv.org/pdf/1801.04381.pdf>)

代码：

```
import numpy as np
import mindspore as ms
import mindspore.nn as nn
import mindspore.ops as ops

def _make_divisible(v, divisor, min_value=None):
    if min_value is None:
        min_value = divisor
    new_v = max(min_value, int(v + divisor / 2) // divisor * divisor)
    # Make sure that round down does not go down by more than 10%.
    if new_v < 0.9 * v:
        new_v += divisor
    return new_v

class GlobalAvgPooling(nn.Cell):

    def __init__(self):
        super(GlobalAvgPooling, self).__init__()
        self.mean = ops.ReduceMean(keep_dims=False)

    def construct(self, x):
        x = self.mean(x, (2, 3))
        return x

class ConvBNReLU(nn.Cell):

    def __init__(self, in_planes, out_planes, kernel_size=3, stride=1, groups=1):
        super(ConvBNReLU, self).__init__()
        padding = (kernel_size - 1) // 2
        in_channels = in_planes
        out_channels = out_planes
        if groups == 1:
            conv = nn.Conv2d(in_channels, out_channels, kernel_size, stride, pad_mode='pad',
padding=padding)
        else:
            out_channels = in_planes
            conv = nn.Conv2d(in_channels, out_channels, kernel_size, stride, pad_mode='pad',
padding=padding, group=in_channels)
```

```
layers = [conv, nn.BatchNorm2d(out_planes), nn.ReLU6()]
self.features = nn.SequentialCell(layers)

def construct(self, x):
    output = self.features(x)
    return output

class InvertedResidual(nn.Cell):

    def __init__(self, inp, oup, stride, expand_ratio):
        super(InvertedResidual, self).__init__()
        assert stride in [1, 2]

        hidden_dim = int(round(inp * expand_ratio))
        self.use_res_connect = stride == 1 and inp == oup

        layers = []
        if expand_ratio != 1:
            layers.append(ConvBNReLU(inp, hidden_dim, kernel_size=1))
        layers.extend([
            # dw
            ConvBNReLU(hidden_dim, hidden_dim,
                        stride=stride, groups=hidden_dim),
            # pw-linear
            nn.Conv2d(hidden_dim, oup, kernel_size=1,
                      stride=1, has_bias=False),
            nn.BatchNorm2d(oup),
        ])
        self.conv = nn.SequentialCell(layers)
        self.add = ops.Add()
        self.cast = ops.Cast()

    def construct(self, x):
        identity = x
        x = self.conv(x)
        if self.use_res_connect:
            return self.add(identity, x)
        return x

class MobileNetV2Backbone(nn.Cell):

    def __init__(self, width_mult=1., inverted_residual_setting=None, round_nearest=8,
                 input_channel=32, last_channel=1280):
        super(MobileNetV2Backbone, self).__init__()
        block = InvertedResidual
        # setting of inverted residual blocks
```

```
self.cfgs = inverted_residual_setting
if inverted_residual_setting is None:
    self.cfgs = [
        # t, c, n, s
        [1, 16, 1, 1],
        [6, 24, 2, 2],
        [6, 32, 3, 2],
        [6, 64, 4, 2],
        [6, 96, 3, 1],
        [6, 160, 3, 2],
        [6, 320, 1, 1],
    ]

# building first layer
input_channel = _make_divisible(input_channel * width_mult, round_nearest)
self.out_channels = _make_divisible(last_channel * max(1.0, width_mult), round_nearest)
features = [ConvBNReLU(3, input_channel, stride=2)]
# building inverted residual blocks
for t, c, n, s in self.cfgs:
    output_channel = _make_divisible(c * width_mult, round_nearest)
    for i in range(n):
        stride = s if i == 0 else 1
        features.append(block(input_channel, output_channel, stride, expand_ratio=t))
        input_channel = output_channel
# building last several layers
features.append(ConvBNReLU(input_channel, self.out_channels, kernel_size=1))
# make it nn.CellList
self.features = nn.SequentialCell(features)
self._initialize_weights()

def construct(self, x):
    x = self.features(x)
    return x

def _initialize_weights(self):

    self.init_parameters_data()
    for _, m in self.cells_and_names():
        if isinstance(m, nn.Conv2d):
            n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
            m.weight.set_data(ms.Tensor(np.random.normal(0, np.sqrt(2. / n),
m.weight.data.shape).astype("float32"))))

            if m.bias is not None:
                m.bias.set_data(
                    ms.numpy.zeros(m.bias.data.shape, dtype="float32"))
            elif isinstance(m, nn.BatchNorm2d):
```

```

        m.gamma.set_data(
            ms.Tensor(np.ones(m.gamma.data.shape, dtype="float32")))
        m.beta.set_data(
            ms.numpy.zeros(m.beta.data.shape, dtype="float32"))

@property
def get_features(self):
    return self.features

class MobileNetV2Head(nn.Cell):

    def __init__(self, input_channel=1280, num_classes=1000, has_dropout=False, activation="None"):
        super(MobileNetV2Head, self).__init__()
        # mobilenet head
        head = ([GlobalAvgPooling()] if not has_dropout else
                 [GlobalAvgPooling(), nn.Dropout(0.2)])
        self.head = nn.SequentialCell(head)
        self.dense = nn.Dense(input_channel, num_classes, has_bias=True)
        self.need_activation = True
        if activation == "Sigmoid":
            self.activation = ops.Sigmoid()
        elif activation == "Softmax":
            self.activation = ops.Softmax()
        else:
            self.need_activation = False
        self._initialize_weights()

    def construct(self, x):
        x = self.head(x)
        x = self.dense(x)
        if self.need_activation:
            x = self.activation(x)
        return x

    def _initialize_weights(self):

        self.init_parameters_data()
        for _, m in self.cells_and_names():
            if isinstance(m, nn.Dense):
                m.weight.set_data(ms.Tensor(np.random.normal(
                    0, 0.01, m.weight.data.shape).astype("float32")))
                if m.bias is not None:
                    m.bias.set_data(
                        ms.numpy.zeros(m.bias.data.shape, dtype="float32"))

class MobileNetV2Combine(nn.Cell):

```

```
def __init__(self, backbone, head):
    super(MobileNetV2Combine, self).__init__(auto_prefix=False)
    self.backbone = backbone
    self.head = head

def construct(self, x):
    x = self.backbone(x)
    x = self.head(x)
    return x

def mobilenet_v2(num_classes):
    backbone_net = MobileNetV2Backbone()
    head_net = MobileNetV2Head(backbone_net.out_channels, num_classes)
    return MobileNetV2Combine(backbone_net, head_net)
```

步骤 4 模型训练与评估

创建好模型，损失函数和优化器后，通过`Model`接口初始化模型后，使用`model.train`接口训练模型，`model.eval`接口评估模型精度。

在本段章节中涉及迁移学习的知识点：

1. 下载预训练模型权重

通过

https://download.mindspore.cn/models/r1.7/mobilenetv2_ascend_v170_imagenet2012_official_cv_top1acc71.88.ckpt 下载已经在 ImageNet 数据集上已经训练好的预训练模型文件，并**存放在运行代码的同级目录下**。

2. 读取预训练模型

通过 `load_checkpoint()` 接口读取预训练模型文件，输出结果为一个字典数据格式。

3. 修改预训练模型参数

修改预训练模型权重相关的参数（预训练模型是在 ImageNet 数据集上训练的 1001 分类任务，而我们当前实验任务是实现 flowers 数据集的 5 分类任务，网络模型修改的是最后一层的全连接层）。

代码：

```
import mindspore
import mindspore.nn as nn
from mindspore.train import Model
from mindspore import Tensor, save_checkpoint
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor
from mindspore.train.serialization import load_checkpoint, load_param_into_net
# 创建模型,其中目标分类数为 5
network = mobilenet_v2(5)

# 加载预训练权重
param_dict =
load_checkpoint("./mobilenetv2_ascend_v170_imagenet2012_official_cv_top1acc71.88.ckpt")
```

```
# 根据修改的模型结构修改相应的权重数据
param_dict["dense.weight"] =
mindspore.Parameter(Tensor(param_dict["dense.weight"][:5, :],mindspore.float32),
name="dense.weight", requires_grad=True)
param_dict["dense.bias"] =
mindspore.Parameter(Tensor(param_dict["dense.bias"][:5, ],mindspore.float32), name="dense.bias",
requires_grad=True)

# 将修改后的权重参数加载到模型中
load_param_into_net(network, param_dict)

train_step_size = dataset_train.get_dataset_size()
epoch_size = 20
lr = nn.cosine_decay_lr(min_lr=0.0, max_lr=0.1,total_step=epoch_size *
train_step_size,step_per_epoch=train_step_size,decay_epoch=200)
#定义优化器
network_opt = nn.Momentum(params=network.trainable_params(), learning_rate=0.01,
momentum=0.9)

# 定义损失函数
network_loss = loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

# 定义评价指标
metrics = {"Accuracy": nn.Accuracy()})

# 初始化模型
model = Model(network, loss_fn=network_loss, optimizer=network_opt, metrics=metrics)

# 损失值监控
loss_cb = LossMonitor(per_print_times=train_step_size)

# 模型保存参数，设置每隔多少步保存一次模型，最多保存几个模型
ckpt_config = CheckpointConfig(save_checkpoint_steps=100, keep_checkpoint_max=10)

# 模型保存，设置模型保存的名称，路径，以及保存参数
ckptpoint_cb = ModelCheckpoint(prefix="mobilenet_v2", directory='./ckpt', config=ckpt_config)

print("===== Starting Training =====")
# 训练模型，设置训练次数为 5，训练集，回调函数
model.train(5, dataset_train, callbacks=[loss_cb,ckptpoint_cb], dataset_sink_mode=True)
# 使用测试集进行模型评估，输出测试集的准确率
metric = model.eval(dataset_val)
print(metric)
```

输出：

```
===== Starting Training =====
```



```
epoch: 1 step: 201, loss is 0.8389087915420532
epoch: 2 step: 201, loss is 0.5519619584083557
epoch: 3 step: 201, loss is 0.26490363478660583
epoch: 4 step: 201, loss is 0.4540162682533264
epoch: 5 step: 201, loss is 0.5963617563247681
{'Accuracy': 0.9166666666666666}
```

步骤 5 可视化模型预测

定义 visualize_model 函数，使用上述验证精度最高的模型对输入图像进行预测，并将预测结果可视化。

代码：

```
import matplotlib.pyplot as plt
import mindspore as ms

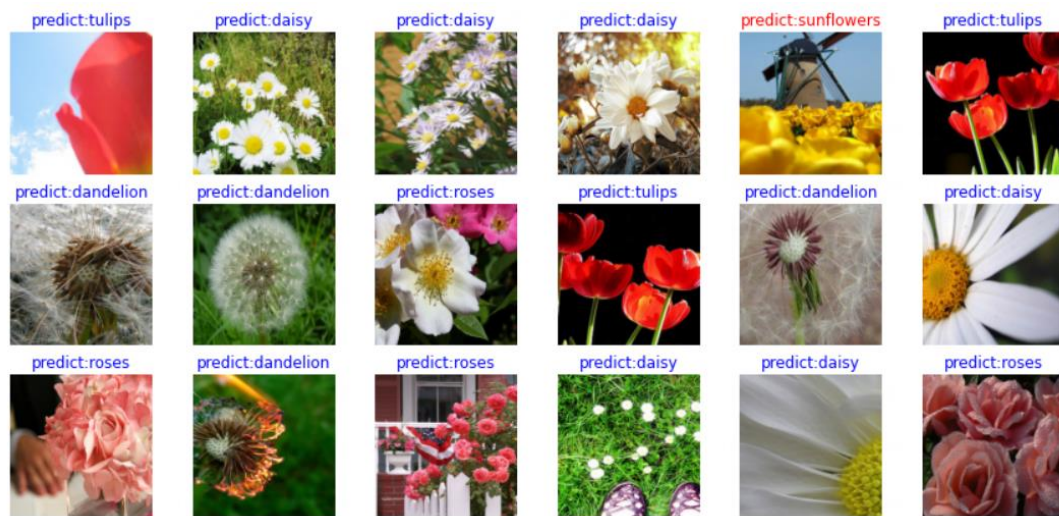
def visualize_model(best_ckpt_path, val_ds):
    num_class = 5 # 对狼和狗图像进行二分类
    net = mobilenet_v2(num_class)
    # 加载模型参数
    param_dict = ms.load_checkpoint(best_ckpt_path)
    ms.load_param_into_net(net, param_dict)
    model = ms.Model(net)
    # 加载验证集的数据进行验证
    data = next(val_ds.create_dict_iterator())
    images = data["image"].asnumpy()
    labels = data["label"].asnumpy()
    class_name = {0:'daisy',1:'dandelion',2:'roses',3:'sunflowers',4:'tulips'}
    # 预测图像类别
    output = model.predict(ms.Tensor(data['image']))
    pred = np.argmax(output.asnumpy(), axis=1)

    # 显示图像及图像的预测值
    plt.figure(figsize=(15, 7))
    for i in range(len(labels)):
        plt.subplot(3, 6, i + 1)
        # 若预测正确，显示为蓝色；若预测错误，显示为红色
        color = 'blue' if pred[i] == labels[i] else 'red'
        plt.title('predict:{}'.format(class_name[pred[i]]), color=color)
        picture_show = np.transpose(images[i], (1, 2, 0))
        mean = np.array([0.485, 0.456, 0.406])
        std = np.array([0.229, 0.224, 0.225])
        picture_show = std * picture_show + mean
        picture_show = np.clip(picture_show, 0, 1)
        plt.imshow(picture_show)
        plt.axis('off')

    plt.show()
```

```
visualize_model('ckpt/mobilenet_v2-5_201.ckpt', dataset_val)
```

输出：



3.4 思考题

通常使用哪个接口读取预训练模型？

4 ResNet50 图像分类实验

4.1 实验介绍

本实验实现的是深度学习领域的经典案例——ResNet50 图像分类实验。整体流程如下：

- 处理需要的数据集，这里使用了花卉图像数据集。
- 定义一个网络，这里自己搭建 ResNet50 模型结构。
- 定义损失函数和优化器。
- 加载数据集并进行训练，训练完成后，利用测试集进行评估。

4.2 实验准备

在动手进行实践之前，确保你已经正确安装了 MindSpore。如果没有，可以通过 MindSpore 官网安装页面：<https://www.mindspore.cn/install/>，将 MindSpore 安装在你的电脑当中。

同时希望你拥有 Python 编码基础和概率、矩阵等基础数学知识。

推荐环境：

版本：MindSpore 1.7

编程语言：Python 3.7

4.3 实验详细设计与实现

4.3.1 数据准备

我们示例中用到的图像花卉数据集，该数据集是开源数据集，总共包括 5 种花的类型：分别是 daisy（雏菊，633 张），dandelion（蒲公英，898 张），roses（玫瑰，641 张），sunflowers（向日葵，699 张），tulips（郁金香，799 张），保存在 5 个文件夹当中，总共 3670 张，大小大概在 230M 左右。为了在模型部署上线之后进行测试，数据集在这里分成了 flower_photos_train 和 flower_photos_test 两部分。

目录结构如下：

flower_photos_train

```
|—— daisy
|—— dandelion
|—— roses
|—— sunflowers
|—— tulips
|—— LICENSE.txt
flower_photos_test
|—— daisy
|—— dandelion
|—— roses
|—— sunflowers
|—— tulips
|—— LICENSE.txt
```

数据集的获取：

```
https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower\_photos\_train.zip
https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower\_photos\_test.zip
```

4.3.2 实验步骤

步骤 1 导入 Python 库&模块并配置运行信息

在使用前，导入需要的 Python 库。

详细的 MindSpore 的模块说明，可以在 MindSpore API 页面中搜索查询。

可以通过 `context.set_context` 来配置运行需要的信息，譬如运行模式、后端信息、硬件等信息。

导入 `context` 模块，配置运行需要的信息。

代码：

```
from easydict import EasyDict as edict
# 字典访问，用来存储超参数
import os
# os 模块主要用于处理文件和目录
import numpy as np
# 科学计算库
import matplotlib.pyplot as plt
# 绘图库
```

```
import mindspore
# MindSpore 库
import mindspore.dataset as ds
# 数据集处理模块
from mindspore.dataset.vision import c_transforms as vision
# 图像增强模块
from mindspore import context
# 环境设置模块
import mindspore.nn as nn
# 神经网络模块
from mindspore.train import Model
# 模型编译
from mindspore.nn.optim.momentum import Momentum
# 动量优化器
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor
# 模型保存设置
from mindspore import Tensor
# 张量
from mindspore.train.serialization import export
# 模型导出
from mindspore.train.loss_scale_manager import FixedLossScaleManager
# 损失值平滑处理
from mindspore.train.serialization import load_checkpoint, load_param_into_net
# 模型加载
import mindspore.ops as ops
# 常见算子操作

# 设置 MindSpore 的执行模式和设备
context.set_context(mode=context.GRAPH_MODE, device_target="CPU")
```

步骤 2 定义参数变量

edict 中存放的是模型训练和测试中所需要的各种参数配置。

代码：

```
cfg = edict({
    'data_path': 'flowers/flower_photos_train',    #训练数据集路径
    'test_path': 'flowers/flower_photos_train',    #测试数据集路径
    'data_size': 3616,
    'HEIGHT': 224, # 图片高度
    'WIDTH': 224, # 图片宽度
    '_R_MEAN': 123.68, # CIFAR10 的均值
    '_G_MEAN': 116.78,
    '_B_MEAN': 103.94,
    '_R_STD': 1, # 自定义的标准差
    '_G_STD': 1,
```

```
'_B_STD':1,
'_RESIZE_SIDE_MIN': 256, # 图像增强 resize 最小值
'_RESIZE_SIDE_MAX': 512,

'batch_size': 32, # 批次大小
'num_class': 5,      # 分类类别
'epoch_size': 5,    # 训练次数
'loss_scale_num':1024,

'prefix': 'resnet-ai', # 模型保存的名称
'directory': './model_resnet', # 模型保存的路径
'save_checkpoint_steps': 10, # 每隔 10 步保存 ckpt
})
```

步骤 3 数据的读取和处理

数据集对于训练非常重要，好的数据集可以有效提高训练精度和效率。在加载数据集前，我们通常会对数据集进行一些处理。

定义数据集及数据操作

我们定义一个函数 create_dataset 来创建数据集。在这个函数中，我们定义好需要进行的数据增强和处理操作：

- 读取数据集。
- 定义进行数据增强和处理所需要的一些参数。
- 根据参数，生成对应的数据增强操作。
- 使用 map 映射函数，将数据操作应用到数据集。
- 对生成的数据集进行处理。
- 对处理好的数据进行样例展示。

代码：

```
# 数据处理
def read_data(path,config,usage="train"):
    # 从目录中读取图像的源数据集。
    dataset = ds.ImageFolderDataset(path,

class_indexing={'daisy':0,'dandelion':1,'roses':2,'sunflowers':3,'tulips':4})
    # define map operations
    # 图像解码算子
    decode_op = vision.Decode()
    # 图像正则化算子
    normalize_op = vision.Normalize(mean=[cfg._R_MEAN, cfg._G_MEAN, cfg._B_MEAN],
std=[cfg._R_STD, cfg._G_STD, cfg._B_STD])
    # 图像调整大小算子
```

```
resize_op = vision.Resize(cfg._RESIZE_SIDE_MIN)
# 图像裁剪算子
center_crop_op = vision.CenterCrop((cfg.HEIGHT, cfg.WIDTH))
# 图像随机水平翻转算子
horizontal_flip_op = vision.RandomHorizontalFlip()
# 图像通道数转换算子
channelswap_op = vision.HWC2CHW()
# 图像随机裁剪解码编码调整大小算子
random_crop_decode_resize_op = vision.RandomCropDecodeResize((cfg.HEIGHT, cfg.WIDTH), (0.5,
1.0), (1.0, 1.0), max_attempts=100)

# 只对训练集做的预处理操作
if usage == 'train':
    dataset = dataset.map(input_columns="image", operations=random_crop_decode_resize_op)
    dataset = dataset.map(input_columns="image", operations=horizontal_flip_op)
# 只对测试集做的预处理操作
else:
    dataset = dataset.map(input_columns="image", operations=decode_op)
    dataset = dataset.map(input_columns="image", operations=resize_op)
    dataset = dataset.map(input_columns="image", operations=center_crop_op)

# 对全部数据集做的预处理操作
dataset = dataset.map(input_columns="image", operations=normalize_op)
dataset = dataset.map(input_columns="image", operations=channelswap_op)

# 对训练集做的批次处理
if usage == 'train':
    dataset = dataset.shuffle(buffer_size=10000) # 10000 as in imageNet train script
    dataset = dataset.batch(cfg.batch_size, drop_remainder=True)
# 对测试集做的批次处理
else:
    dataset = dataset.batch(1, drop_remainder=True)

# 数据增强
dataset = dataset.repeat(1)

dataset.map_model = 4

return dataset

# 查看训练集和测试集的数量
de_train = read_data(cfg.data_path, cfg, usage="train")
de_test = read_data(cfg.test_path, cfg, usage="test")
print('训练数据集数量: ', de_train.get_dataset_size()*cfg.batch_size) # get_dataset_size() 获取批处理的大小。
print('测试数据集数量: ', de_test.get_dataset_size())
```

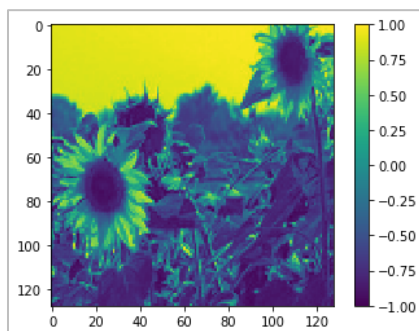
```
# 查看训练集的样图
data_next = de_train.create_dict_iterator(output_numpy=True).__next__()
print('通道数/图像长/宽: ', data_next['image'][0,...].shape)
print('一张图像的标签样式: ', data_next['label'][0]) # 一共 5 类, 用 0-4 的数字表达类别。

plt.figure()
plt.imshow(data_next['image'][0,0,...])
plt.colorbar()
plt.grid(False)
plt.show()
```

输出:

```
训练数据集数量: 3616
测试数据集数量: 32
通道数/图像长/宽: (3, 224, 224)
一张图像的标签样式: 3
```

数据处理展示:



步骤 4 模型构建训练

● 定义模型

残差块 (Residual Block)

将前面若干层的输出跳过中间层作为后几层的输入部分, 也就是说后面特征层会有前几层的部分线性贡献。此种设计是为了克服随着网络层数加深而产生的学习效率变低和准确率无法有效提升的问题。

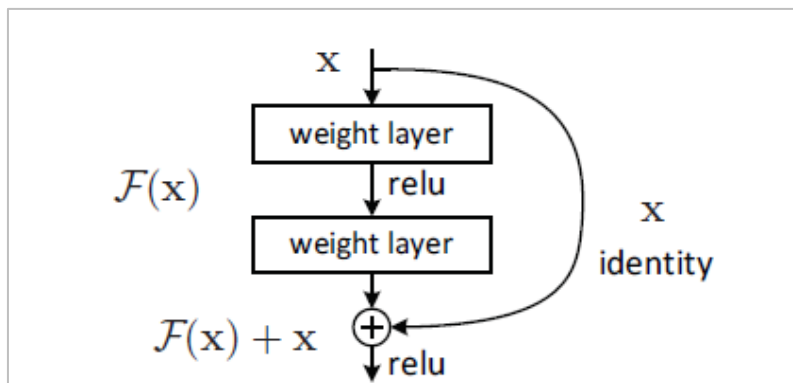


图4-1 残差块

如果维度相同：

$$y = F(x, W_i) + x$$

$$F = W_2 \sigma(W_1 x)$$

如果维度不同，则采用：

$$y = F(x, W_i) + W_s x$$

瓶颈（BottleNeck）模块：

瓶颈（BottleNeck）模块，思路和 Inception 模块一样，通过 1x1 conv 来巧妙地缩减或扩张 feature map 维度从而使 3x3 conv 的 filters 数目不受外界即上一层输入的影响，自然它的输出也不会影响到下一层 module。

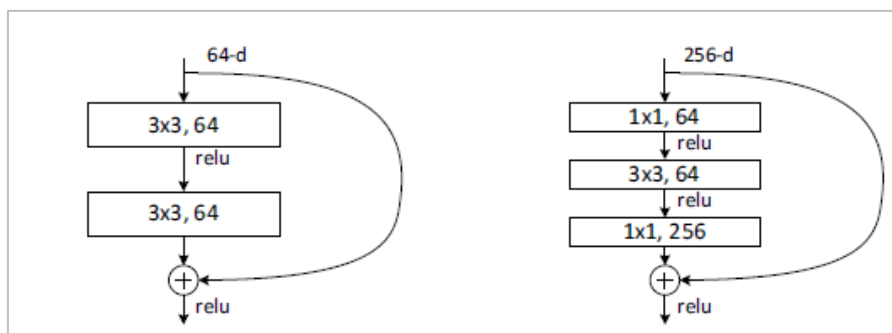


图4-2 BottleNeck

ResNet50 模型构建

ResNet50 有两个基本的块，分别名为 Conv Block 和 Identity Block，其中 Conv Block 输入和输出的维度是不一样的，所以不能连续串联，它的作用是改变网络的维度；Identity Block 输入维度和输出维度相同，可以串联，用于加深网络。

代码：

```
"""ResNet."""

# 定义权重初始化函数
def _weight_variable(shape, factor=0.01):
    init_value = np.random.randn(*shape).astype(np.float32) * factor
```

```
return Tensor(init_value)

# 定义 3X3 卷积函数
def _conv3x3(in_channel, out_channel, stride=1):
    weight_shape = (out_channel, in_channel, 3, 3)
    weight = _weight_variable(weight_shape)
    return nn.Conv2d(in_channel, out_channel,
                     kernel_size=3, stride=stride, padding=0, pad_mode='same', weight_init=weight)

# 定义 1X1 卷积层函数
def _conv1x1(in_channel, out_channel, stride=1):
    weight_shape = (out_channel, in_channel, 1, 1)
    weight = _weight_variable(weight_shape)
    return nn.Conv2d(in_channel, out_channel,
                     kernel_size=1, stride=stride, padding=0, pad_mode='same', weight_init=weight)

# 定义 7X7 卷积层函数
def _conv7x7(in_channel, out_channel, stride=1):
    weight_shape = (out_channel, in_channel, 7, 7)
    weight = _weight_variable(weight_shape)
    return nn.Conv2d(in_channel, out_channel,
                     kernel_size=7, stride=stride, padding=0, pad_mode='same', weight_init=weight)

# 定义 Batch Norm 层函数
def _bn(channel):
    return nn.BatchNorm2d(channel, eps=1e-4, momentum=0.9,
                          gamma_init=1, beta_init=0, moving_mean_init=0, moving_var_init=1)

# 定义最后一层的 Batch Norm 函数
def _bn_last(channel):
    return nn.BatchNorm2d(channel, eps=1e-4, momentum=0.9,
                          gamma_init=0, beta_init=0, moving_mean_init=0, moving_var_init=1)

# 定义全连接层函数
def _fc(in_channel, out_channel):
    weight_shape = (out_channel, in_channel)
    weight = _weight_variable(weight_shape)
    return nn.Dense(in_channel, out_channel, has_bias=True, weight_init=weight, bias_init=0)

# 构建残差模块
class ResidualBlock(nn.Cell):
    """
    ResNet V1 residual block definition.

    Args:
        in_channel (int): Input channel.
        out_channel (int): Output channel.
```

stride (int): Stride size for the first convolutional layer. Default: 1.

Returns:

Tensor, output tensor.

Examples:

```
>>> ResidualBlock(3, 256, stride=2)
```

```
.....
```

expansion = 4 # conv2_x--conv5_x 中，前两层的卷积核的个数是第三层（也就是输出通道）的 4 分之一。

```
def __init__(self, in_channel, out_channel, stride=1):
    super(ResidualBlock, self).__init__()
```

```
# 前两层的卷积核个数等于输出通道的四分之一
channel = out_channel // self.expansion
```

```
# 第一层卷积
```

```
self.conv1 = _conv1x1(in_channel, channel, stride=1)
self.bn1 = _bn(channel)
```

```
# 第二层卷积
```

```
self.conv2 = _conv3x3(channel, channel, stride=stride)
self.bn2 = _bn(channel)
```

```
# 第三层卷积，其中卷积核个数等于输出通道
```

```
self.conv3 = _conv1x1(channel, out_channel, stride=1)
self.bn3 = _bn_last(out_channel)
```

```
# Relu 激活层
```

```
self.relu = nn.ReLU()
```

```
self.down_sample = False
```

```
# 当步长不为 1、或输出通道不等于输入通道时，进行图像下采样，用来调整通道数
```

```
if stride != 1 or in_channel != out_channel:
```

```
    self.down_sample = True
```

```
self.down_sample_layer = None
```

```
# 用 1X1 卷积调整通道数
```

```
if self.down_sample:
```

```
    self.down_sample_layer = nn.SequentialCell([_conv1x1(in_channel, out_channel, stride), #
1X1 卷积
```

```
        _bn(out_channel)]) # Batch Norm
```

```
# 加法算子
```

```
self.add = ops.Add()
```

```
# 构建残差块
```

```
def construct(self, x):
```

```
# 输入
identity = x

# 第一层卷积 1X1
out = self.conv1(x)
out = self.bn1(out)
out = self.relu(out)

# 第二层卷积 3X3
out = self.conv2(out)
out = self.bn2(out)
out = self.relu(out)

# 第三层卷积 1X1
out = self.conv3(out)
out = self.bn3(out)

# 改变网络的维度
if self.down_sample:
    identity = self.down_sample_layer(identity)

# 加上残差
out = self.add(out, identity)
# Relu 激活
out = self.relu(out)

return out

# 构建残差网络
class ResNet(nn.Cell):
    """
    ResNet architecture.

    Args:
        block (Cell): Block for network.
        layer_nums (list): Numbers of block in different layers.
        in_channels (list): Input channel in each layer.
        out_channels (list): Output channel in each layer.
        strides (list): Stride size in each layer.
        num_classes (int): The number of classes that the training images are belonging to.

    Returns:
        Tensor, output tensor.

    Examples:
        >>> ResNet(ResidualBlock,
        >>>         [3, 4, 6, 3],
        >>>         [64, 256, 512, 1024],
```

```
>>> [256, 512, 1024, 2048],
>>> [1, 2, 2, 2],
>>> 10)
.....

# 输入参数为：残差块，残差块重复数，输入通道，输出通道，步长，图像类别数
def __init__(self, block, layer_nums, in_channels, out_channels, strides, num_classes):
    super(ResNet, self).__init__()
    if not len(layer_nums) == len(in_channels) == len(out_channels) == 4:
        raise ValueError("the length of layer_num, in_channels, out_channels list must be 4!")

    # 第一层卷积，卷积核 7X7，输入通道 3，输出通道 64，步长 2
    self.conv1 = _conv7x7(3, 64, stride=2)
    self.bn1 = _bn(64)
    self.relu = ops.ReLU()

    # 3X3 池化层，步长 2
    self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, pad_mode="same")

    # conv2_x 残差块
    self.layer1 = self._make_layer(block,
                                    layer_nums[0],
                                    in_channel=in_channels[0],
                                    out_channel=out_channels[0],
                                    stride=strides[0])

    # conv3_x 残差块
    self.layer2 = self._make_layer(block,
                                    layer_nums[1],
                                    in_channel=in_channels[1],
                                    out_channel=out_channels[1],
                                    stride=strides[1])

    # conv4_x 残差块
    self.layer3 = self._make_layer(block,
                                    layer_nums[2],
                                    in_channel=in_channels[2],
                                    out_channel=out_channels[2],
                                    stride=strides[2])

    # conv5_x 残差块
    self.layer4 = self._make_layer(block,
                                    layer_nums[3],
                                    in_channel=in_channels[3],
                                    out_channel=out_channels[3],
                                    stride=strides[3])

    # 均值算子
    self.mean = ops.ReduceMean(keep_dims=True)
    # Flatten 层
    self.flatten = nn.Flatten()
```

```
# 输出层
self.end_point = _fc(out_channels[3], num_classes)

# 输入参数为：残差块，残差块重复数，输入通道，输出通道，步长
def _make_layer(self, block, layer_num, in_channel, out_channel, stride):
    """
    Make stage network of ResNet.

    Args:
        block (Cell): Resnet block.
        layer_num (int): Layer number.
        in_channel (int): Input channel.
        out_channel (int): Output channel.
        stride (int): Stride size for the first convolutional layer.

    Returns:
        SequentialCell, the output layer.

    Examples:
        >>> _make_layer(ResidualBlock, 3, 128, 256, 2)
    """
    # 搭建 convn_x 的残差块

    layers = []

    resnet_block = block(in_channel, out_channel, stride=stride)
    layers.append(resnet_block)

    for _ in range(1, layer_num):
        resnet_block = block(out_channel, out_channel, stride=1)
        layers.append(resnet_block)

    return nn.SequentialCell(layers)

# 构建 ResNet 网络
def construct(self, x):
    x = self.conv1(x) # 第一层卷积 7X7，步长为 2
    x = self.bn1(x)   # 第一层的 Batch Norm
    x = self.relu(x)  # Rule 激活层
    c1 = self.maxpool(x) # 最大池化 3X3，步长为 2

    c2 = self.layer1(c1) # conv2_x 残差块
    c3 = self.layer2(c2) # conv3_x 残差块
    c4 = self.layer3(c3) # conv4_x 残差块
    c5 = self.layer4(c4) # conv5_x 残差块
```

```

        out = self.mean(c5, (2, 3)) # 平均池化层
        out = self.flatten(out) # Flatten 层
        out = self.end_point(out) # 输出层

    return out

# 构建 ResNet50 网络
def resnet50(class_num=5):
    """
    Get ResNet50 neural network.

    Args:
        class_num (int): Class number.

    Returns:
        Cell, cell instance of ResNet50 neural network.

    Examples:
        >>> net = resnet50(10)
    """
    return ResNet(ResidualBlock, # 残差块
                  [3, 4, 6, 3], # 残差块数量
                  [64, 256, 512, 1024], # 输入通道
                  [256, 512, 1024, 2048], # 输出通道
                  [1, 2, 2, 2], # 步长
                  class_num) # 输出类别数

```

● 开始训练

完成数据预处理、网络定义、损失函数和优化器定义之后，开始模型训练。模型训练包含 2 层迭代，数据集的多轮迭代 epoch 和每一轮迭代按分组从数据集中抽取数据，输入网络计算得到损失函数，然后通过优化器计算和更新训练参数的梯度。

1. 下载预训练模型

新建目录 model_resnet，通过

https://download.mindspore.cn/models/r1.7/resnet50_ascend_v170_imagenet2012_official_cv_top1acc76.97_top5acc93.44.ckpt 下载已经在 ImageNet 数据集上已经训练好的预训练模型文件，并存放在指定目录 model_resnet 下。

2. 加载预训练模型

通过 load_checkpoint() 接口读取预训练模型文件获取字典格式的参数文件。

3. 修改预训练模型参数

修改预训练模型参数的最后一层连接参数（预训练模型是在 ImageNet 数据集上训练的 1001 分类任务，而我们当前实验任务是实现 flowers 数据集的 5 分类任务）。因此我们需要修改最后一层全连接层的参数。

代码：

```
# 构建 ResNet50 网络，输出类别数为 5，对应 5 种花的类别
net=resnet50(class_num=cfg.num_class)

#读取预训练模型参数
param_dict =
load_checkpoint("model_resnet/resnet50_ascend_v170_imagenet2012_official_cv_top1acc76.97_top5acc9
3.44.ckpt")

#展示读取的模型参数
print(param_dict)

#通过 mindspore.Parameter ( ) 修改 end_point.weight 和 end_point.bias 对应的 shape
param_dict["end_point.weight"] = mindspore.Parameter(Tensor(param_dict["end_point.weight"][:5, :],
mindspore.float32), name="variable")
param_dict["end_point.bias"] = mindspore.Parameter(Tensor(param_dict["end_point.bias"][:5,],
mindspore.float32), name="variable")

# 设置 Softmax 交叉熵损失函数
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

# 设置学习率
train_step_size = de_train.get_dataset_size()
lr = nn.cosine_decay_lr(min_lr=0.0001, max_lr=0.001, total_step=train_step_size *
cfg.epoch_size,step_per_epoch=train_step_size, decay_epoch=cfg.epoch_size)

# 设置动量优化器
opt = Momentum(net.trainable_params(), lr, momentum=0.9, weight_decay=1e-4,
loss_scale=cfg.loss_scale_num)

# 损失值平滑，解决训练过程中梯度过小的问题
loss_scale = FixedLossScaleManager(cfg.loss_scale_num, False)

# 模型编译，输入网络结构，损失函数，优化器，损失值平滑，以及模型评估标准
model = Model(net, loss_fn=loss, optimizer=opt, loss_scale_manager=loss_scale, metrics={'acc'})

# 损失值监控
loss_cb = LossMonitor(per_print_times=train_step_size)

# 模型保存参数，设置每隔多少步保存一次模型，最多保存几个模型
ckpt_config = CheckpointConfig(save_checkpoint_steps=cfg.save_checkpoint_steps,
keep_checkpoint_max=1)

# 模型保存，设置模型保存的名称，路径，以及保存参数
ckptpoint_cb = ModelCheckpoint(prefix=cfg.prefix, directory=cfg.directory, config=ckpt_config)

print("===== Starting Training =====")
# 训练模型，设置训练次数，训练集，回调函数，是否采用数据下沉模式（只可应用于 Ascend 和 GPU，可
加快训练速度）
```



```
model.train(cfg.epoch_size, de_train, callbacks=[loss_cb, ckpoint_cb], dataset_sink_mode=True)
# 训练时长大约 15-20 分钟

# 使用测试集进行模型评估，输出测试集的准确率
metric = model.eval(de_test)
print(metric)
```

输出：

```
===== Starting Training =====
epoch: 1 step: 113, loss is 0.23505911231040955
epoch: 2 step: 113, loss is 0.11479882150888443
epoch: 3 step: 113, loss is 0.13273288309574127
epoch: 4 step: 113, loss is 0.42304447293281555
epoch: 5 step: 113, loss is 0.14625898003578186
{'acc': 0.9587912087912088}
```

步骤 5 模型预测

通过训练好的权重文件，调用 `model.predict()` 对测试数据进行测试，输出预测结果和真实结果。

代码：

```
# 模型预测，从测试集中取 10 个样本进行测试，输出预测结果和真实结果
class_names = {0:'daisy',1:'dandelion',2:'roses',3:'sunflowers',4:'tulips'}
for i in range(10):
    test_ = de_test.create_dict_iterator().__next__()
    test = Tensor(test_['image'], mindspore.float32)
    # 模型预测
    predictions = model.predict(test)
    predictions = predictions.asnumpy()
    true_label = test_['label'].asnumpy()
    # 显示预测结果
    p_np = predictions[0, :]
    pre_label = np.argmax(p_np)
    print('第' + str(i) + '个 sample 预测结果: ', class_names[pre_label], ' 真实结果: ',
          class_names[true_label[0]])
```

输出：

```
第 0 个 sample 预测结果: sunflowers  真实结果: sunflowers
第 1 个 sample 预测结果: daisy      真实结果: daisy
第 2 个 sample 预测结果: roses      真实结果: roses
第 3 个 sample 预测结果: roses      真实结果: roses
第 4 个 sample 预测结果: roses      真实结果: roses
第 5 个 sample 预测结果: dandelion   真实结果: dandelion
第 6 个 sample 预测结果: dandelion   真实结果: dandelion
第 7 个 sample 预测结果: roses      真实结果: roses
```

第 8 个 sample 预测结果: tulips 真实结果: tulips
第 9 个 sample 预测结果: dandelion 真实结果: dandelion

4.4 思考题

本实验中用到了 ResNet50 网络，该网络共有多少层（只计算卷积与全连接层）？并指出本实验中对模型训练定义了多少个 Epoch？网络的输出类别数是多少？

5 TextCNN 情感分析实验

5.1 实验介绍

本实验实现的是深度学习领域的经典案例——TextCNN 情感分析实验。整体流程如下：

- 下载需要的数据集，这里使用了 rt-polarity 数据集，定义数据预处理函数。
- 生成数据用于训练和验证。
- 定义 TextCNN 模型结构搭建、训练、评估、加载离线模型、在线推理函数。
- 定义训练需要的各种参数优化器、损失函数、保存检查点、时间监视等。
- 加载数据集并进行训练，训练完成后，利用测试集进行评估。

5.2 实验准备

在动手进行实践之前，确保你已经正确安装了 MindSpore。如果没有，可以通过 MindSpore 官网安装页面：<https://www.mindspore.cn/install/>，将 MindSpore 安装在你的电脑当中。

同时希望你拥有 Python 编码基础和概率、矩阵等基础数学知识。

推荐环境：

版本：MindSpore 1.7

编程语言：Python 3.7

5.3 实验详细设计与实现

5.3.1 数据准备

实例中使用的 rt-polarity，是电影英文评论数据，rt-polarity.pos 里面存储的是 5000 电影英文好评，rt-polarity.neg 里面存的是 5000 条英文版差评。

下载链接：<https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V3.5/chapter4/TextCNN.zip>

数据目录结构如下：

└─data

```
|— rt-polarity.pos
|— rt-polarity.neg
```

5.3.2 实验步骤

步骤 1 导入 Python 库&模块并配置运行信息

在使用前，导入需要的 Python 库。

目前使用到 os 库，为方便理解，其他需要的库，我们在具体使用到时再说明。详细的 MindSpore 的模块说明，可以在 MindSpore API 页面中搜索查询。可以通过 context.set_context 来配置运行需要的信息，譬如运行模式、后端信息、硬件等信息。

导入 context 模块，配置运行需要的信息。

代码：

```
import math
import numpy as np
import pandas as pd
import os
import math
import random
import codecs
from pathlib import Path

import mindspore
import mindspore.dataset as ds
import mindspore.nn as nn
from mindspore import Tensor
from mindspore import context
from mindspore.train.model import Model
from mindspore.nn.metrics import Accuracy
from mindspore.train.serialization import load_checkpoint, load_param_into_net
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor
from mindspore.ops import operations as ops

from easydict import EasyDict as edict

cfg = edict({
    'name': 'movie review',
    'pre_trained': False,
    'num_classes': 2,
    'batch_size': 64,
    'epoch_size': 4,
    'weight_decay': 3e-5,
    'data_path': './data/',
    'device_target': 'CPU',
    'device_id': 0,
```

```
'keep_checkpoint_max': 1,
'checkpoint_path': './ckpt/train_textcnn-4_149.ckpt',
'word_len': 51,
'vec_length': 40
})

context.set_context(mode=context.GRAPH_MODE, device_target=cfg.device_target,
device_id=cfg.device_id)
```

本实验中我们的运行模式是图模式。根据实际情况配置硬件信息，譬如代码运行在 Ascend AI 处理器上，则 device_target 选择 Ascend，代码运行在 CPU、GPU 同理。详细参数说明，请参见 context.set_context 接口说明（https://www.mindspore.cn/docs/zh-CN/r1.7/api_python/mindspore.context.html?highlight=context）。

步骤 2 数据读取和处理

数据预览代码：

```
with open("./data/rt-polarity.neg", 'r', encoding='utf-8') as f:
    print("Negative reivevs:")
    for i in range(5):
        print("[{0}]:{1}".format(i,f.readline()))
with open("./data/rt-polarity.pos", 'r', encoding='utf-8') as f:
    print("Positive reivevs:")
    for i in range(5):
        print("[{0}]:{1}".format(i,f.readline()))
```

数据预览输出：

```
Negative reivevs:
[0]:simplistic , silly and tedious .

[1]:it's so laddish and juvenile , only teenage boys could possibly find it funny .

[2]:exploitative and largely devoid of the depth or sophistication that would make watching such a graphic treatment of the crimes bearable .

[3]:[garbus] discards the potential for pathological study , exhuming instead , the skewed melodrama of the circumstantial situation .

[4]:a visually flashy but narratively opaque and emotionally vapid exercise in style and mystification .

Positive reivevs:
[0]:the rock is destined to be the 21st century's new " conan " and that he's going to make a splash even greater than arnold schwarzenegger , jean-claud van damme or stev
en segal .

[1]:the gorgeously elaborate continuation of " the lord of the rings " trilogy is so huge that a column of words cannot adequately describe co-writer/director peter jackso
n's expanded vision of j . r . r . tolkien's middle-earth .

[2]:effective but too-tepid biopic

[3]:if you sometimes like to go to the movies to have fun , wasabi is a good place to start .

[4]:emerges as something rare , an issue movie that's so honest and keenly observed that it doesn't feel like one .
```

定义数据处理函数代码：

```
#定义数据生成类
class Generator():
    def __init__(self, input_list):
        self.input_list=input_list
    def __getitem__(self,item):
        return (np.array(self.input_list[item][0],dtype=np.int32),
                np.array(self.input_list[item][1],dtype=np.int32))
    def __len__(self):
        return len(self.input_list)
```

```
class MovieReview:
    """
    影评数据集
    """
    def __init__(self, root_dir, maxlen, split):
        """
        input:
            root_dir: 影评数据目录
            maxlen: 设置句子最大长度
            split: 设置数据集中训练/评估的比例
        """
        self.path = root_dir
        self.feelMap = {
            'neg':0,
            'pos':1
        }
        self.files = []

        self.doConvert = False

        mypath = Path(self.path)
        if not mypath.exists() or not mypath.is_dir():
            print("please check the root_dir!")
            raise ValueError

        # 在数据目录中找到文件
        for root,_,filename in os.walk(self.path):
            for each in filename:
                self.files.append(os.path.join(root,each))
            break

        # 确认是否为两个文件.neg 与.pos
        if len(self.files) != 2:
            print("There are {} files in the root_dir".format(len(self.files)))
            raise ValueError

        # 读取数据
        self.word_num = 0
        self.maxlen = 0
        self.minlen = float("inf")
        self.maxlen = float("-inf")
        self.Pos = []
        self.Neg = []
        for filename in self.files:
```

```
self.read_data(filename)

self.text2vec(maxlen=maxlen)
self.split_dataset(split=split)

def read_data(self, filePath):
    with open(filePath, 'r') as f:
        for sentence in f.readlines():
            sentence = sentence.replace("\n", "")\
                .replace("'", "")\
                .replace('"', "")\
                .replace('.', '')\
                .replace(',', '')\
                .replace('[', '')\
                .replace(']', '')\
                .replace('(', '')\
                .replace(')', '')\
                .replace(':', '')\
                .replace('--', '')\
                .replace('-', '')\
                .replace('\\', '')\
                .replace('0', '')\
                .replace('1', '')\
                .replace('2', '')\
                .replace('3', '')\
                .replace('4', '')\
                .replace('5', '')\
                .replace('6', '')\
                .replace('7', '')\
                .replace('8', '')\
                .replace('9', '')\
                .replace(' ', '')\
                .replace('=', '')\
                .replace('$', '')\
                .replace('/', '')\
                .replace('*', '')\
                .replace(';', '')\
                .replace('<b>', '')\
                .replace('%', '')

            sentence = sentence.split(' ')
            sentence = list(filter(lambda x: x, sentence))
            if sentence:
                self.word_num += len(sentence)
                self.maxlen = self.maxlen if self.maxlen >= len(sentence) else len(sentence)
                self.minlen = self.minlen if self.minlen <= len(sentence) else len(sentence)
            if 'pos' in filePath:
```

```
        self.Pos.append([sentence,self.feelMap['pos']])
    else:
        self.Neg.append([sentence,self.feelMap['neg']])

def text2vec(self, maxlen):
    """
    # 将句子转化为向量
    """
    # Vocab = {word : index}
    self.Vocab = dict()

    # self.Vocab['None']
    for SentenceLabel in self.Pos+self.Neg:
        vector = [0]*maxlen
        for index, word in enumerate(SentenceLabel[0]):
            if index >= maxlen:
                break
            if word not in self.Vocab.keys():
                self.Vocab[word] = len(self.Vocab)
                vector[index] = len(self.Vocab) - 1
            else:
                vector[index] = self.Vocab[word]
        SentenceLabel[0] = vector
    self.doConvert = True

def split_dataset(self, split):
    """
    分割为训练集和测试集
    """
    trunk_pos_size = math.ceil((1-split)*len(self.Pos))
    trunk_neg_size = math.ceil((1-split)*len(self.Neg))
    trunk_num = int(1/(1-split))
    pos_temp=list()
    neg_temp=list()
    for index in range(trunk_num):
        pos_temp.append(self.Pos[index*trunk_pos_size:(index+1)*trunk_pos_size])
        neg_temp.append(self.Neg[index*trunk_neg_size:(index+1)*trunk_neg_size])
    self.test = pos_temp.pop(2)+neg_temp.pop(2)
    self.train = [i for item in pos_temp+neg_temp for i in item]

random.shuffle(self.train)

def get_dict_len(self):
    """
    获得数据集中文字组成的词典长度
    """
    if self.doConvert:
```



```

        return len(self.Vocab)
    else:
        print("Haven't finished Text2Vec")
        return -1

    def create_train_dataset(self, epoch_size, batch_size):
        dataset = ds.GeneratorDataset(
            source=Generator(input_list=self.train),
            column_names=["data", "label"],
            shuffle=False
        )
        dataset=dataset.batch(batch_size=batch_size,drop_remainder=True)
        dataset=dataset.repeat(epoch_size)
        return dataset

    def create_test_dataset(self, batch_size):
        dataset = ds.GeneratorDataset(
            source=Generator(input_list=self.test),
            column_names=["data", "label"],
            shuffle=False
        )
        dataset=dataset.batch(batch_size=batch_size,drop_remainder=True)
        return dataset

```

通过自定义 read_data 函数加载原始数据集后使用 text2vec 函数对数据进行文本进行向量化处理。使用 split_dataset 将数据分割为训练数据和测试数据，然后 create_test_dataset 和 create_train_dataset 调用 mindspore.dataset.GeneratorDataset 生成数据来进行训练和验证。

config 配置代码：

```

instance = MovieReview(root_dir=cfg.data_path, maxlen=cfg.word_len, split=0.9)
dataset = instance.create_train_dataset(batch_size=cfg.batch_size,epoch_size=cfg.epoch_size)
batch_num = dataset.get_dataset_size()

```

显示数据处理结果代码：

```

vocab_size=instance.get_dict_len()
print("vocab_size:{0}".format(vocab_size))
item =dataset.create_dict_iterator()
for i,data in enumerate(item):
    if i<1:
        print(data)
        print(data['data'][1])
    else:
        break

```

输出数据处理结果：

[illegible]

步骤 3 配置训练参数

学习率设置代码:

```
learning_rate = []
warm_up = [1e-3 / math.floor(cfg.epoch_size / 5) * (i + 1) for _ in range(batch_num)
            for i in range(math.floor(cfg.epoch_size / 5))]
shrink = [1e-3 / (16 * (i + 1)) for _ in range(batch_num)
           for i in range(math.floor(cfg.epoch_size * 3 / 5))]
normal_run = [1e-3 for _ in range(batch_num) for i in
               range(cfg.epoch_size - math.floor(cfg.epoch_size / 5)
                     - math.floor(cfg.epoch_size * 2 / 5))]
learning_rate = learning_rate + warm_up + normal_run + shrink
```

通过 epoch 实现训练中的动态学习率。

步骤 4 TextCNN 模型定义

模型类定义了模型结构搭建、训练、评估、加载离线模型、在线推理函数。

代码:

```
def _weight_variable(shape, factor=0.01):
    init_value = np.random.randn(*shape).astype(np.float32) * factor
    return Tensor(init_value)

def make_conv_layer(kernel_size):
    weight_shape = (96, 1, *kernel_size)
    weight = _weight_variable(weight_shape)
    return nn.Conv2d(in_channels=1, out_channels=96, kernel_size=kernel_size, padding=1,
                     pad_mode="pad", weight_init=weight, has_bias=True)

class TextCNN(nn.Cell):
    def __init__(self, vocab_len, word_len, num_classes, vec_length):
        super(TextCNN, self).__init__()
        self.vec_length = vec_length
        self.word_len = word_len
        self.num_classes = num_classes

        self.unsqueeze = ops.ExpandDims()
        self.embedding = nn.Embedding(vocab_len, self.vec_length, embedding_table='normal')
```

```
self.slice = ops.Slice()
self.layer1 = self.make_layer(kernel_height=3)
self.layer2 = self.make_layer(kernel_height=4)
self.layer3 = self.make_layer(kernel_height=5)

self.concat = ops.Concat(1)

self.fc = nn.Dense(96*3, self.num_classes)
self.drop = nn.Dropout(keep_prob=0.5)
self.print = ops.Print()
self.reducemean = ops.ReduceMax(keep_dims=False)

def make_layer(self, kernel_height):
    return nn.SequentialCell(
        [
            make_conv_layer((kernel_height,self.vec_length)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(self.word_len-kernel_height+1,1)),
        ]
    )

def construct(self,x):
    x = self.unsqueeze(x, 1)
    x = self.embedding(x)
    x1 = self.layer1(x)
    x2 = self.layer2(x)
    x3 = self.layer3(x)

    x1 = self.reducemean(x1, (2, 3))
    x2 = self.reducemean(x2, (2, 3))
    x3 = self.reducemean(x3, (2, 3))

    x = self.concat((x1, x2, x3))
    x = self.drop(x)
    x = self.fc(x)
    return x

net = TextCNN(vocab_len=instance.get_dict_len(), word_len=cfg.word_len,
              num_classes=cfg.num_classes, vec_length=cfg.vec_length)
print(net)
```

查看神经网络结构：

```
TextCNN(
  (embedding): Embedding(vocab_size=18848, embedding_size=40, use_one_hot=False, embedding_table=Parameter (name=embedding.embedding_table), dtype=Float32, padding_idx=None)
)
(layer1): SequentialCell<
  (0): Conv2d(input_channels=1, output_channels=96, kernel_size=(3, 40), stride=(1, 1), pad_mode=pad, padding=1, dilation=(1, 1), group=1, has_bias=True, weight_init=[[[[
1.47466036e-02 -8.85174982e-03 6.20904262e-04 ... 5.97969582e-03
6.87394897e-03 -4.35171695e-03]
[ 1.25721507e-02 1.11018270e-02 3.73373111e-03 ... -1.32954121e-02
1.67019237e-02 6.70977589e-03]
[ 5.16406354e-03 6.65343972e-03 -9.12646390e-03 ... 4.12355090e-04
6.96751568e-03 2.65645944e-02]]]]
[ [[ [ 2.20983545e-03 9.51934140e-03 5.41285099e-03 ... -2.79155909e-03
-9.01016127e-03 1.36778364e-02]
[ -9.36647598e-03 -1.73871801e-03 1.08188950e-02 ... 1.51296274e-03
7.57558912e-04 5.44300769e-03]
[ 1.13182161e-02 4.26522718e-04 1.49796065e-03 ... -3.42557859e-03
-1.79516233e-03 -4.79884632e-03]]]]
[ [[ [-4.98654880e-03 -2.08284725e-02 2.18074489e-02 ... -7.44788814e-03
-4.96819383e-03 1.28578511e-03]
[ 1.82666897e-03 1.37834558e-02 -4.40165540e-03 ... -8.17192066e-03
9.86121874e-03 1.46957850e-02]
[ 1.46543132e-02 7.15820771e-03 5.12730749e-03 ... -1.95071772e-02
-1.01376360e-03 6.32047653e-03]]]]

```

步骤 5 定义训练的相关参数

损失函数：又叫目标函数，用于衡量预测值与实际值差异的程度。深度学习通过不停地迭代来缩小损失值。定义一个好的损失函数，可以有效提高模型的性能。

优化器：用于最小化损失函数，从而在训练过程中改进模型。

定义了损失函数后，可以得到损失函数关于权重的梯度。梯度用于指示优化器优化权重的方向，以提高模型性能。MindSpore 支持的损失函数有 SoftmaxCrossEntropyWithLogits、L1Loss、MSELoss 等。这里使用 SoftmaxCrossEntropyWithLogits 损失函数。

MindSpore 提供了 Callback 机制，可以在训练过程中执行自定义逻辑，这里以框架提供的 ModelCheckpoint 为例。ModelCheckpoint 可以保存网络模型和参数，以便进行后续的 fine-tuning（微调）操作。

代码：

```
# 优化器、损失函数、保存检查点、时间监视器等设置
opt = nn.Adam(filter(lambda x: x.requires_grad, net.get_parameters()),
               learning_rate=learning_rate, weight_decay=cfg.weight_decay)
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
model = Model(net, loss_fn=loss, optimizer=opt, metrics={'acc': Accuracy()})
config_ck = CheckpointConfig(save_checkpoint_steps=int(cfg.epoch_size*batch_num/2),
                             keep_checkpoint_max=cfg.keep_checkpoint_max)
time_cb = TimeMonitor(data_size=batch_num)
ckpt_save_dir = "/ckpt"
ckptpoint_cb = ModelCheckpoint(prefix="train_textcnn", directory=ckpt_save_dir, config=config_ck)
loss_cb = LossMonitor()
```

步骤 6 启动训练

训练过程指的是将训练集数据传入网络对网络中的参数进行训练优化的过程。在 MindSpore 框架中 Model.train 方法用于完成这一过程。

代码：

```
model.train(cfg.epoch_size, dataset, callbacks=[time_cb, ckptpoint_cb, loss_cb])
print("train success")
```

输出：

```
epoch: 1 step: 596, loss is 0.023292765
epoch time: 36818.071 ms, per step time: 61.775 ms
epoch: 2 step: 596, loss is 0.0016317479
epoch time: 4320.621 ms, per step time: 7.249 ms
epoch: 3 step: 596, loss is 0.00025401893
epoch time: 4271.667 ms, per step time: 7.167 ms
epoch: 4 step: 596, loss is 0.0001774891
epoch time: 4332.598 ms, per step time: 7.269 ms
train success
```

步骤 7 测试评估

本步骤使用取单条评论文本数据，进行测试，输出情感类别及其概率。

代码：

```
def preprocess(sentence):
    sentence = sentence.lower().strip()
    sentence = sentence.replace('\n','')\
        .replace(' ','')\
        .replace('\','')\
        .replace('.',',')\
        .replace(',',',')\
        .replace('[','')\
        .replace(']',',')\
        .replace('(','')\
        .replace(')',',')\
        .replace(':',',')\
        .replace('--','')\
        .replace('-',',')\
        .replace('\\','')\
        .replace('0','')\
        .replace('1','')\
        .replace('2','')\
        .replace('3','')\
        .replace('4','')\
        .replace('5','')\
        .replace('6','')\
        .replace('7','')\
        .replace('8','')\
        .replace('9','')\
        .replace('','')\
        .replace('=','')\
        .replace('$','')\
        .replace('/',',')\
        .replace('*','')\
        .replace(';','')\
        .replace('<b>','')
```

```
                .replace('%','')\
                .replace(" ", " ")

sentence = sentence.split(' ')
maxlen = cfg.word_len
vector = [0]*maxlen
for index, word in enumerate(sentence):
    if index >= maxlen:
        break
    if word not in instance.Vocab.keys():
        print(word,"单词未出现在字典中")
    else:
        vector[index] = instance.Vocab[word]
sentence = vector

return sentence

def inference(review_en):
    review_en = preprocess(review_en)
    input_en = Tensor(np.array([review_en]).astype(np.int32))
    output = net(input_en)
    if np.argmax(np.array(output[0])) == 1:
        print("Positive comments")
    else:
        print("Negative comments")
```

代码：

```
review_en = "the movie is so boring"
inference(review_en)
```

输出：

Negative comments

5.4 思考题

如果想通过 GPU 跑以上实验，需要在哪个地方修改相关配置为 GPU？

华为认证 HCIA-AI 系列教程

HCIA-AI人工智能

综合实验指导手册

版本：3.5



华为技术有限公司

版权所有 © 华为技术有限公司 2022。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技术架构与应用认证、云服务与平台认证两类认证。

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。

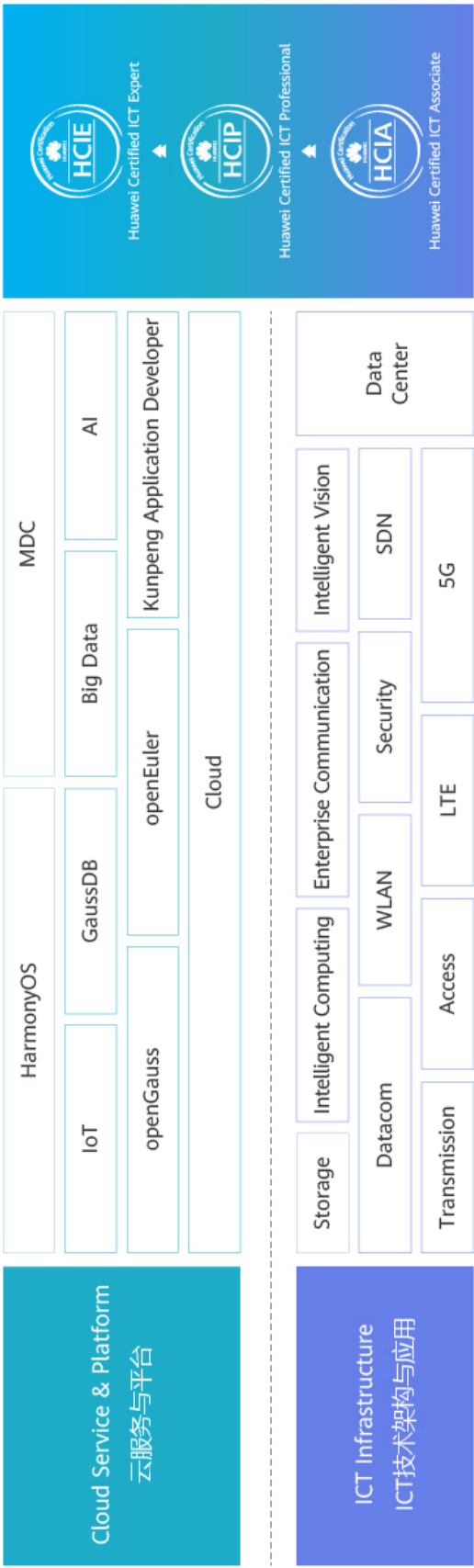
华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

华为认证HCIA-AI V3.5定位于培养和认证具备使用机器学习、深度学习等算法进行AI产品和解决方案设计、开发和创新能力的工程师。

通过HCIA-AI V3.5认证，将证明您了解人工智能发展历史、华为昇腾AI体系和全栈全场景AI战略知识，掌握传统机器学习和深度学习的相关算法；具备利用MindSpore开发框架和MindSpore开发框架进行搭建、训练、部署神经网络的能力；能够胜任人工智能领域销售、市场、产品经理、项目管理、技术支持等岗位。

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！

华为认证



前言

简介

本书为 HCIA-AI 认证培训教程，适用于准备参加 HCIA-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，您将能够了解利用华为云实现 ModelArts 自动学习、图像识别（图像标签）、文字识别（通用文字识别、通用表格识别、手写文字识别）、语音合成和语音识别功能开发。

内容描述

本实验指导书共包含 4 组实验，涉及到基于 Python 语言的华为云 ModelArts 自动学习、图像识别、文字识别、语音合成及识别内容，帮助学员提升 AI 的实践开发能力。

- 实验 1 掌握如何使用华为云的 ModelArts 自动学习服务。
- 实验 2 掌握如何使用华为云的图像识别服务。
- 实验 3 掌握如何使用华为云的文字识别服务。
- 实验 4 掌握如何使用华为云的语音合成和语音识别服务。

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 语言编程能力，会使用 Jupyter Notebook。
- 有一定的计算机操作及计算机网络基础，能够理解华为云服务返回的状态信息。

实验环境说明

- Python3.7, Jupyter Notebook。
 - 实验需要访问网络，请保持网络连接。
-

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 自动学习	3
1.1 实验简介	3
1.2 实验目的	3
1.3 实验环境说明	3
1.4 花卉识别应用实验步骤	4
1.4.2 创建项目	5
1.4.3 数据标注	6
1.4.4 模型训练	7
1.4.5 部署预测	8
2 图像识别	10
2.1 实验简介	10
2.2 实验目的	10
2.3 实验准备	11
2.3.1 环境准备	11
2.3.2 SDK 及包安装	11
2.4 实验步骤	12
2.5 实验小结	15
3 文字识别	16
3.1 实验简介	16
3.2 实验目的	16
3.3 实验准备	16
3.3.1 环境准备	16
3.3.2 SDK 安装	17
3.4 实验步骤	17
3.5 思考题	21

3.6 实验小结	21
4 语音合成和语音识别	22
4.1 实验简介	22
4.2 实验目的	22
4.3 实验准备	22
4.3.1 环境准备	22
4.3.2 SDK 获取和配置	23
4.4 实验步骤	23
4.4.1 语音合成	23
4.4.2 语音识别	25
4.5 实验小结	26

1 自动学习

1.1 实验简介

自动学习是 ModelArts 提供的一项服务，可以根据标注数据自动设计模型、自动调参、自动训练、自动压缩和部署模型，不需要代码编写和模型开发经验，即可实现零基础构建 AI 模型。本实验将指导学员完成图片分类，物体检测以及预测分析三种场景的学习和使用。

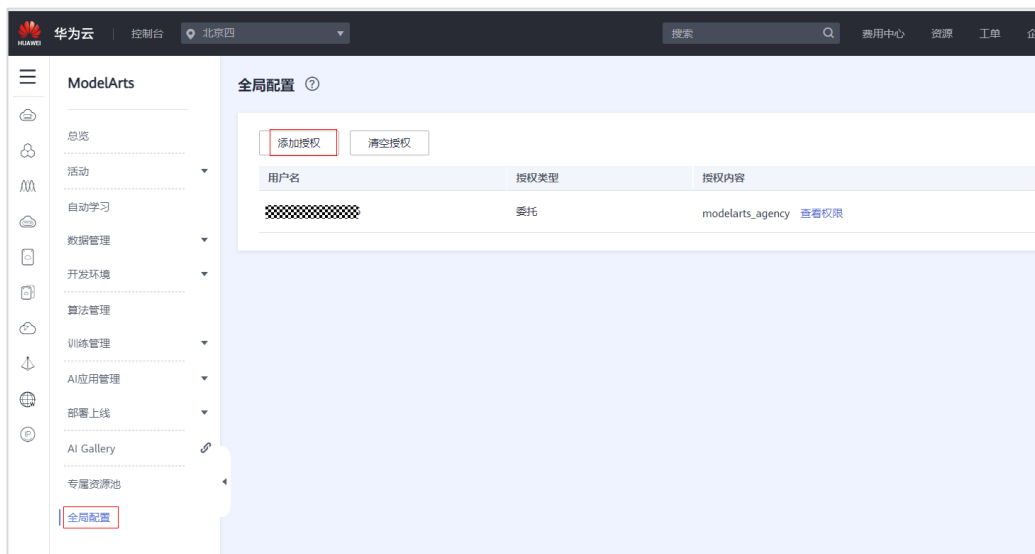
图像分类主要是基于图像的内容进行标记，模型可以预测出图像所对应的标签，适用于图片类别明显的场景。

1.2 实验目的

本实验通过具体的例子来帮助学员快速的创建图像分类模型。花卉识别实验，用于识别图片中花的类别，学员可以快速的了解到图像分类的场景以及使用方法。

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：



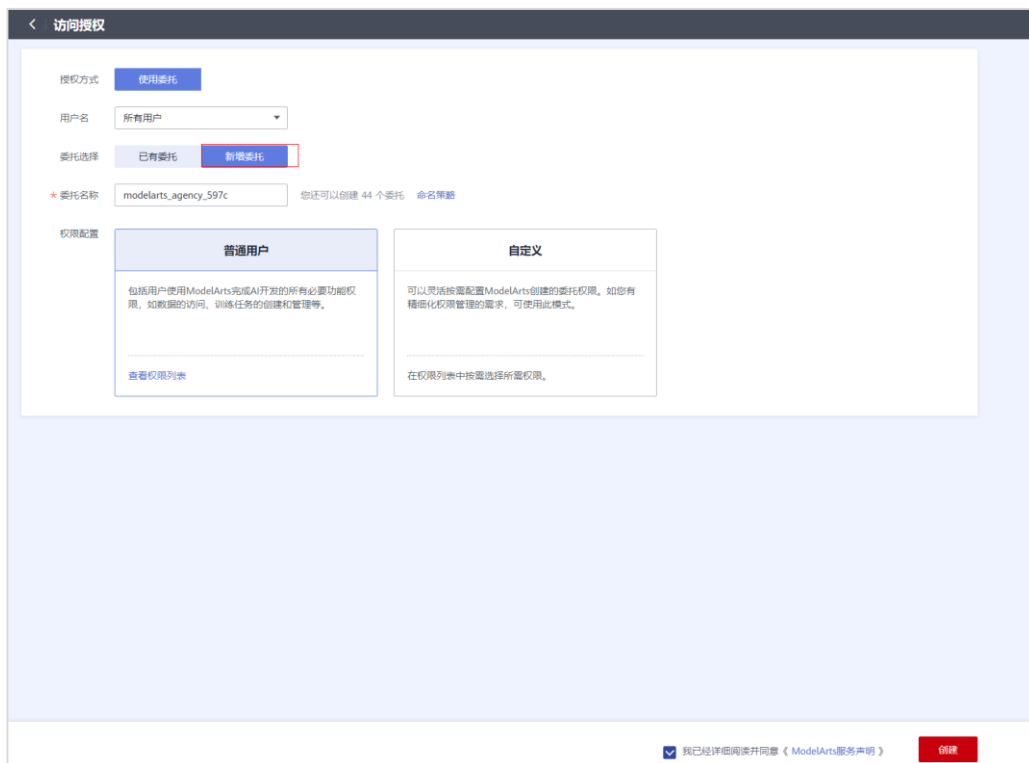



图1-1 ModelArts 操作台

1.4 花卉识别应用实验步骤

ModelArts 自动学习界面主要分为 2 部分，界面上方列举了当前支持的自动学习项目类型，单击“创建项目”可创建自动学习项目；界面下方为已创建的自动学习项目列表，您可以在列表右上方根据项目的类型进行过滤显示，或者在文本框中输入名称，单击 进行查询。

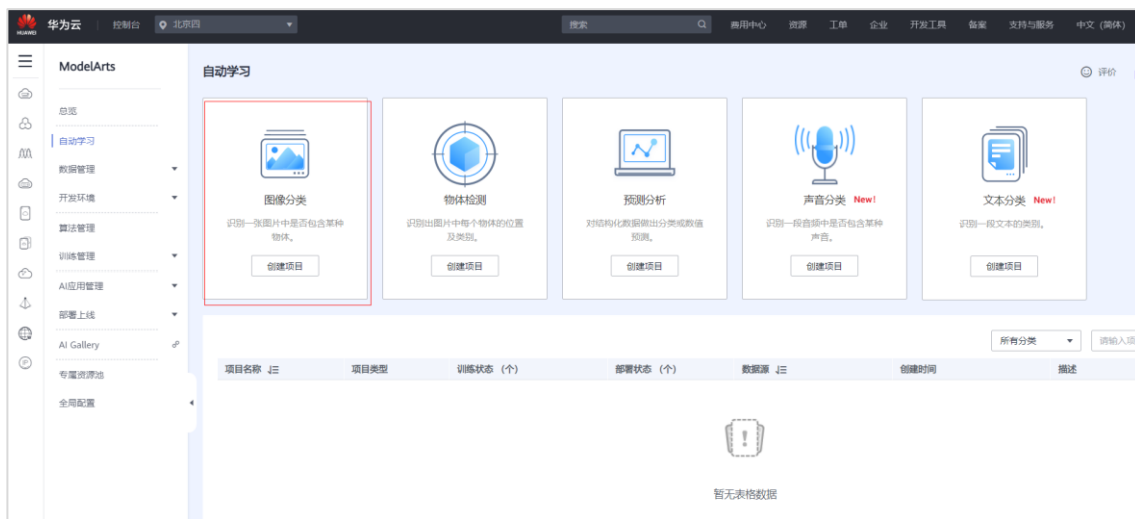


图1-2 自动学习界面

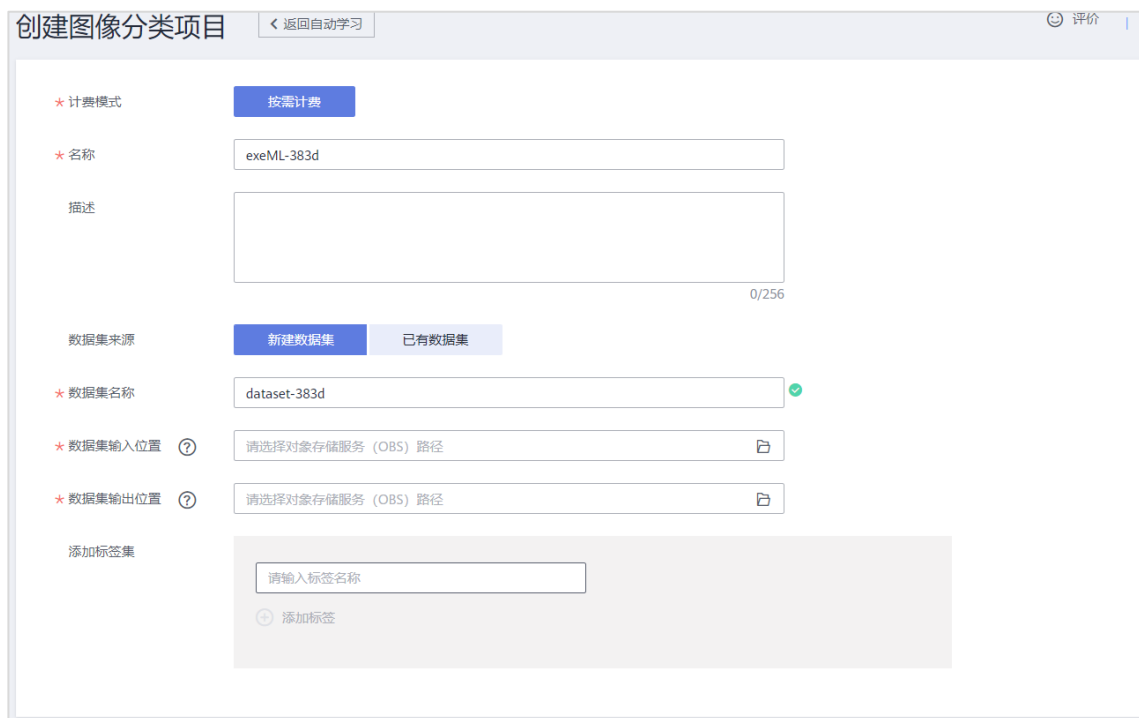
使用流程分为 4 部分，分别是：

- 创建项目：使用 ModelArts 自动学习前，首先需要创建一个自动学习项目。
- 数据标注：上传图片，并进行类别标注。
- 模型训练：数据标注完成后，开始训练模型。
- 部署预测：部署模型，在线预测。

1.4.1 创建项目

步骤 1 项目创建

单击“自动学习”界面中的图像分类“创建项目”，弹出“创建项目”对话框，如图所示。



The screenshot shows the 'Create Image Classification Project' dialog box. It includes a 'Return to Auto Learning' button at the top left. The main form has several sections: 'Billing Mode' with a 'Pay as you go' button; 'Name' field with 'exeML-383d'; 'Description' field; 'Dataset Source' with 'New Dataset' and 'Existing Dataset' buttons; 'Dataset Name' field with 'dataset-383d'; 'Dataset Input Location' and 'Dataset Output Location' fields, both with a question mark icon and a folder icon; and a 'Add Labels' section with a text input field and a 'Add Label' button.

图1-3 创建项目

参数说明：

计费方式：默认为按需计费。

名称：可随意填写。

数据集来源：选择数据集所在的 OBS 路径。首先在 OBS 中新建空文件夹（点击桶名进入桶，点击“新建文件夹”，输入文件夹名称，点击“确定”按钮。）。路径选择新建的 OBS 文件夹路径。（或者可提前将数据导入 OBS，本例中将以下数据链接下载解压后 train 文件夹内数据上传至 OBS/ExeML/flower/train 目录下。

上传数据方式请参考：

https://support.huaweicloud.com/modelarts_faq/modelarts_05_0013.html

数据链接：

<https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V3.5/chapter5/ExeML.zip>

步骤 2 创建完成

点击“创建项目”，完成自动学习项目的创建。

1.4.2 数据标注

步骤 1 图片上传

自动学习项目创建完成后会自动跳转到数据标注界面，会自动加载创建项目中配置的 OBS 目录中已上传图片，如果有已经标注完成的图片，将在已标注区域显示，如下图所示。

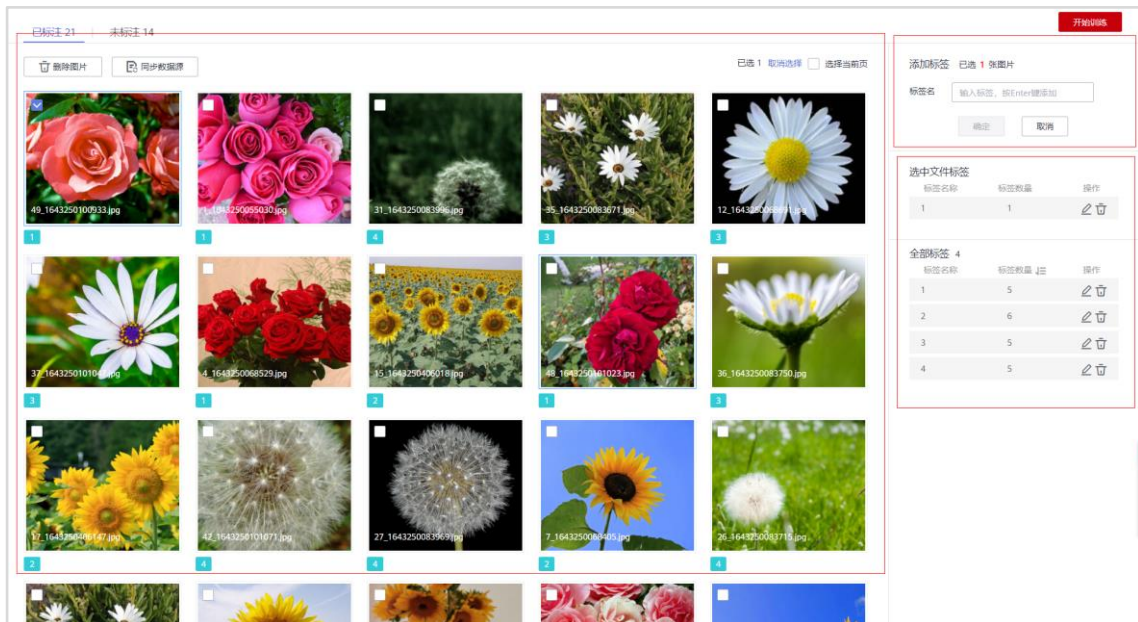


图1-4 图像分类数据标注界面

注意：

- 用于训练的图片，至少有 2 种以上的分类，每种分类的图片数不少于 5 张，即至少有 2 种类型标签且每个标签图片数量不少于 5 张。本例中将该数据集分 tulips、daisy、sunflowers、roses、dandelion 五个标签。
- 图片标注支持多标签，即一张图片可添加多个标签。

步骤 2 图片标注

选择未标注的图片。单击左侧区域“未标注”，然后单击未标注图片依次选中图片，或勾选右上方“选择当前页”选中该页面所有图片。选中图片后，在右上方区域输入新的标签名称或从弹出的列表中选择已添加的标签，然后按 Enter 键添加，如图 1-5 所示。单击“确定”，完成选中图片的标注操作。



图1-5 图像分类图片标注

步骤 3 标签修改

删除或修改单个图片标签。单击左侧区域中的“已标注”，然后单击选中图片，然后在右侧标签名中输入新的标签名称，点击确定即可（可选中多张，进行批量修改）。

1.4.3 模型训练

完成图片标注后，可进行模型的训练。开始训练之前，需要设置训练参数，然后再发布训练，即开始模型的自动训练。由于用于训练的图片，至少有 2 种以上的分类，每种分类的图片数不少于 5 张。因此在发布训练之前，请确保已标注的图片符合要求，否则“开始训练”会处于灰色状态。

步骤 1 参数设置

参数可使用默认值，或者修改训练时长，以及打开高级设置，设置推理时间。训练设置如图所示。

×

训练设置

数据集版本名称	<input style="width: 90%;" type="text" value="V001"/>
训练验证比例 ?	<div style="display: flex; justify-content: space-between;"> <div> 训练集比例: <input style="width: 80%;" type="text" value="0.8"/> ? 验证集比例: 0.2 </div> </div>
增量训练版本 ?	<div style="border: 1px solid #ccc; padding: 2px 5px; display: flex; justify-content: space-between; align-items: center;">不选择版本 ▼</div>
最大训练时长 (分钟)	<input style="width: 90%;" type="text" value="10"/>
训练偏好 ?	<div style="border: 1px solid #ccc; padding: 2px 5px; display: flex; justify-content: space-between; align-items: center;">balance ▼</div>
计算规格	<div style="border: 1px solid #ccc; padding: 2px 5px; display: flex; justify-content: space-between; align-items: center;">增强计算型1实例-自动学习 (GPU) ▼</div>

配置费用 ¥20.00/小时 ?

下一步

图1-6 训练设置

参数说明：

最大训练时长：在该时长内若训练还未完成，则强制退出。为防止训练中退出，建议使用较大值。

步骤 2 模型训练

参数填写完成后点击“开始训练”，模型开始训练。训练结束后，可在图像分类“模型训练”界面查看模型训练结果。

1.4.4 部署预测

步骤 1 部署上线

完成模型训练后，可选择准确率理想且训练状态为“运行成功”的版本部署上线。单击“模型训练”界面版本管理中的“部署”，即可完成模型的部署操作。部署完成后，您也可以在“部署上线>在线服务”查看部署的服务。部署服务如图所示。

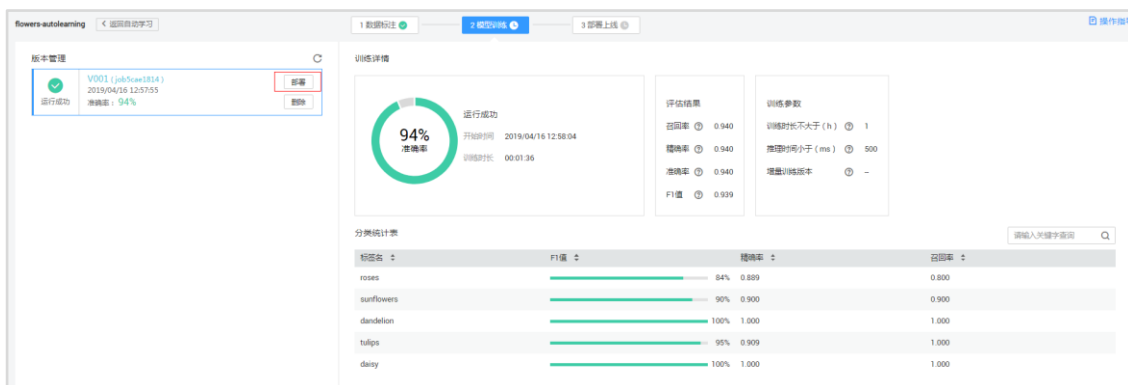


图1-7 部署上线

步骤 2 服务测试

模型部署完成后，您可添加图片进行测试。测试数据的路径是“OBS/flower/test/daisy.jpg”，在“部署上线”界面，单击图片选择按钮，然后选择图片。图片上传成功后，单击“测试”即可进行服务的测试，右侧会显示测试结果，如图所示。其中数据标注中有 5 类标签：tulips、daisy、sunflowers、roses、dandelion，添加的测试图片为雏菊，即 daisy。测试结果“daisy”评分最高，即分类结果为 daisy。

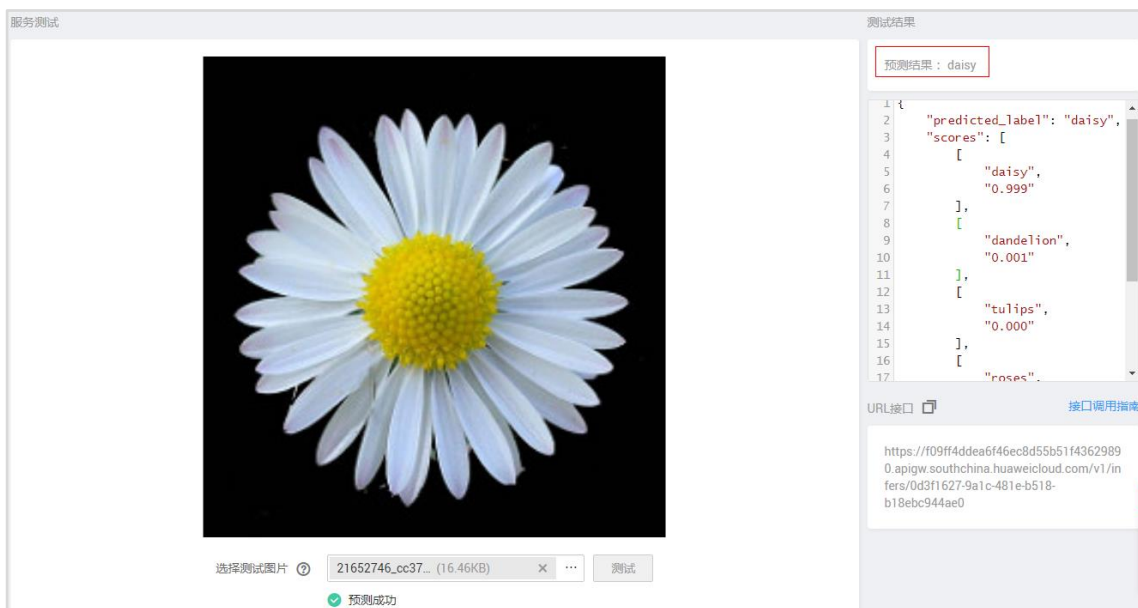


图1-8 服务测试

2 图像识别

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，帮助用户自动处理关键数据，打造智能化业务系统，提升业务效率。

图像识别（Image Recognition），基于深度学习技术，可准确识别图像中的视觉内容，提供多种物体、场景和概念标签，具备目标检测和属性识别等能力，帮助客户准确识别和理解图像内容。

图像识别-主体识别服务是利用华为云图像识别后台算法来检测用户传入图像中的主体内容并返回图片中主体的坐标信息。

图像识别-图像标签服务，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，图像标签服务准确识别自然图片中数百种场景、上千种通用物体及其属性，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直观。使用该功能接口时，用户发送图像，后台算法返回图片标签内容及相应置信度。

图像识别服务通常可以通过以下两种方式进行调用，一种是调用华为云提供的 SDK，另一种是调用相应服务的 API 接口。用户首先需要在华为云“EI 企业智能>人工智能>图像识别 Image”页面开通相关服务（服务只需要开通一次即可，后面使用时无需再申请）。其次，利用 API 接口的开发者可以在华为云提供的 [API Explorer](#) 对不同服务的接口进行调测，同时 API Explorer 上提供有多种编程语言的 SDK 代码示例可供开发者参考使用。

2.2 实验目的

本实验主要介绍了使用华为云图像识别服务，通过本实验学员将了解如何利用华为云的图像识别服务进行图像标签功能。目前华为云有提供基于 Python 语言的图像识别 SDK，本实验将指导学员理解和掌握如何使用 Python 进行图像标签业务的开发方法和技巧。

2.3 实验准备

2.3.1 环境准备

1. 注册并登录华为云管理控制台。
2. 了解图像识别 SDK 相关文档，详见 https://support.huaweicloud.com/sdkreference-image/image_04_0011.html。
3. 开通图像识别服务：登录图像识别管理控制台（https://console.huaweicloud.com/image_recognition/?region=cn-north-4），依次选择左侧的“服务列表”，“图像标签/媒资图像标签”，分别在界面单击“开通”。服务开通一次即可，后续使用时无需再开通。相关服务资费详情请参考华为云价格计算器。
4. 准备华为云账号的AK/SK。如果之前可以获取过，可以继续使用之前的AK/SK。如果之前没有生成过AK/SK，可登录华为云，在用户名处点击“我的凭证”，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。之后的实验不用再新增，可以直接使用此AK/SK信息。
5. 准备project_id。如果之前已经获取过，还可以继续使用之前的project_id。如果没有获取过，可在“我的凭证”界面的项目列表中查看项目ID，复制需要使用区域的项目ID为自己的project_id。

项目ID	项目	所属区域
07693c7eba000fe22f9	cn-north-4	华北-北京四
07691909f080266a2f4	cn-east-2	华东-上海二

6. 已经安装好 Python 环境，Python SDK 适用于 Python3。

2.3.2 SDK 及包安装

请确认已安装 Python 包管理工具 pip，并确认安装好 setuptools，requests 和 websocket-client，在 Jupyter Notebook 界面可通过“!pip list”（在 Jupyter Notebook 的 ipynb 记事本 Cell 中键入“!”表示调用系统命令）查看已安装包列表。如果没有安装，可以在系统终端中激活 Jupyter Notebook 所使用的虚拟环境并执行以下命令安装：

```

pip install setuptools
pip install requests
pip install websocket-client
pip install opencv-python
pip install bounding-box
pip install huaweicloudsdkimage
    
```

2.4 实验步骤

该实验需要安装华为公有云服务提供的图像识别 SDK，通过调用 SDK 底层接口服务并通过 AK/SK 信息进行用户身份认证。本实验就是通过 SDK 来调用图像识别的服务，并在 Jupyter Notebook 中实验，具体步骤如下：

步骤 1 导入所需要的包

```
# coding: utf-8

from huaweicloudsdkcore.auth.credentials import BasicCredentials
from huaweicloudsdkimage.v2.region.image_region import ImageRegion
from huaweicloudsdkcore.exceptions import exceptions
from huaweicloudsdkimage.v2 import *
```

步骤 2 定义图像转 Base64 函数

```
def encode_to_base64(filename):
    imgstr = ""
    with open(filename, 'rb') as file:
        imgstr = base64.b64encode(file.read())
    return imgstr
```

步骤 3 配置用户认证及调用服务区域的相关参数

```
# 此处需要输出您的 AK/SK 信息
ak = "****" #配置自己的 ak
sk = "****" #配置自己的 sk
credentials = BasicCredentials(ak, sk)

client = ImageClient.new_builder() \
    .with_credentials(credentials) \
    .with_region(ImageRegion.value_of("cn-north-4")) \
    .build()
```

步骤 4 执行图像标签任务

```
try:
    request = RunImageTaggingRequest()
    request.body = ImageTaggingReq(
        # 此处替换为公网可以访问的图片地址
        url = "https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V3.5/chapter5/8e1ea806c2ab42668e426e05d1e51c29.png"
    )
    response = client.run_image_tagging(request)
    print(response)
except exceptions.ClientRequestException as e:
    print(e.status_code)
    print(e.request_id)
```



```
print(e.error_code)
print(e.error_msg)
```

输出结果：

```
{
  "result": {
    "tags": [
      {
        "confidence": "97.98",
        "type": "学习办公类",
        "tag": "书",
        "i18n_tag": {
          "zh": "书",
          "en": "Book"
        },
        "i18n_type": {
          "zh": "学习办公类",
          "en": "Learning/office category"
        },
        "instances": [
          {
            "bounding_box": {
              "height": 192.25198877774753,
              "top_left_x": 119.66677770247826,
              "top_left_y": 373.5925991351788,
              "width": 245.83168695523187,
              "confidence": "44.59"
            },
            "confidence": "89.29",
            "type": "人",
            "tag": "人",
            "i18n_tag": {
              "zh": "人",
              "en": "Person"
            },
            "i18n_type": {
              "zh": "人",
              "en": "People"
            },
            "instances": []
          },
          {
            "confidence": "74.08",
            "type": "学习办公类",
            "tag": "海报",
            "i18n_tag": {
              "zh": "海报",
              "en": "Poster"
            },
            "i18n_type": {
              "zh": "学习办公类",
              "en": "Learning/office category"
            },
            "instances": []
          },
          {
            "confidence": "70.85",
            "type": "教育",
            "tag": "报纸",
            "i18n_tag": {
              "zh": "报纸",
              "en": "Newspaper"
            },
            "i18n_type": {
              "zh": "教育",
              "en": "Education"
            },
            "instances": []
          },
          {
            "confidence": "69.15",
            "type": "家居类",
            "tag": "餐桌",
            "i18n_tag": {
              "zh": "餐桌",
              "en": "Table"
            },
            "i18n_type": {
              "zh": "家居类",
              "en": "Home category"
            },
            "instances": []
          }
        ]
      }
    ]
  }
}
```

步骤 5 下载 url 图像

可在 Jupyter Notebook 中输入以下命令下载测试链接 url 图片并保存为 test.png，也可以手动下载图像并保存至与 notebook 同级目录。

```
!curl https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V3.5/chapter5/8e1ea806c2ab42668e426e05d1e51c29.png -o test.png
```



图2-1 test.png

步骤 6 对本地图像进行图像标签任务（选做）

try:


```
request = RunImageTaggingRequest()
request.body = ImageTaggingReq(
    limit = 3,
    image = encode_to_base64('test.png')
)
response = client.run_image_tagging(request)
print(response)
except exceptions.ClientRequestException as e:
    print(e.status_code)
    print(e.request_id)
    print(e.error_code)
    print(e.error_msg)
```

ImageTaggingReq 函数 body 部分可填入 image、url 等相关参数。

image: 要求为图像数据，为 base64 编码，要求 base64 编码后大小不超过 10M，最短边至少 15px，最长边最大 4096px，支持 JPG/PNG/BMP 格式。

url: 此项与 image 参数二选一，支持公网 HTTP/HTTPS URL，华为云 OBS 的 URL。

language: 可选“zh”或“en”，即返回标签的语言类型。默认值为 zh。

threshold: 置信度阈值 0~100，低于此置信度的标签将不会返回，默认为 60。

limit: 最多返回的 tag 数，默认为 50。

输出结果（代码中已 limit 输出结果为 3）：

```
{
  "result": {
    "tags": [
      {
        "confidence": "97.98",
        "type": "学习办公类",
        "tag": "书",
        "i18n_tag": {
          "zh": "书",
          "en": "Book"
        },
        "i18n_type": {
          "zh": "学习办公类",
          "en": "Learning/office category"
        },
        "instances": [
          {
            "bounding_box": {
              "height": 192.25198877774753,
              "top_left_x": 119.66677770247826,
              "top_left_y": 373.5925991351788,
              "width": 245.83168695523187
            },
            "confidence": "44.59"
          }
        ]
      },
      {
        "confidence": "89.29",
        "type": "人",
        "tag": "人",
        "i18n_tag": {
          "zh": "人",
          "en": "Person"
        },
        "i18n_type": {
          "zh": "人",
          "en": "People"
        },
        "instances": []
      },
      {
        "confidence": "74.08",
        "type": "学习办公类",
        "tag": "海报",
        "i18n_tag": {
          "zh": "海报",
          "en": "Poster"
        },
        "i18n_type": {
          "zh": "学习办公类",
          "en": "Learning/office category"
        },
        "instances": []
      }
    ]
  }
}
```

返回结果有 3 个标签，tags 代表标签列表集合，confidence 代表置信度，i18n_tag 代表标签的国际化字段（zh 对应中文，en 对应英文），tag 代表标签名称，type 代表标签类别。

步骤 7 从返回结果中提取 bonding_box 边界信息

```
import json
result = json.loads(str(response))
label = result['result']['tags'][0]['i18n_tag']['en']
height = result['result']['tags'][0]['instances'][0]['bounding_box']['height']
top_left_x = result['result']['tags'][0]['instances'][0]['bounding_box']['top_left_x']
top_left_y = result['result']['tags'][0]['instances'][0]['bounding_box']['top_left_y']
width = result['result']['tags'][0]['instances'][0]['bounding_box']['width']
```

label: 标签信息

width: 检测框区域宽度

height: 检测框区域高度

top_left_x: 检测框左上角到垂直轴距离

top_left_y: 检测框左上角到水平轴距离

步骤 8 给图像添加 bounding_box 并保存

```
from bounding_box import bounding_box as bb
import cv2
image=cv2.imread('test.png')
bb.add(image, top_left_x, top_left_y, top_left_x + width, top_left_y + height, label, 'yellow')
cv2.imwrite("result.png", image)
```



图2-2 result.png

2.5 实验小结

本章主要介绍了应用华为公有云上的图像标签检测服务的具体操作，主要是通过配置用户的认证信息及准备好所需检测材料信息并调用 SDK 进行相关检测服务，并利用返回的结果信息对原始图像进行加工。本章中主要针对用户 AK/SK 认证的方式进行了系统的介绍和说明，为使用图像标签服务提供了实际的操作指导。

3 文字识别

3.1 实验简介

文字识别（Optical Character Recognition，OCR）是指将图片、扫描件或 PDF、OFD 文档中的打印字符进行检测识别成可编辑的文本格式。OCR 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。

通用文字识别：提取图片内的文字及其对应位置信息，并能够根据文字在图片中的位置进行结构化整理工作。

通用表格识别：提取表格内的文字和所在行列位置信息，适应不同格式的表格。同时也识别表格外部的文字区域。用于各种单据和报表的电子化，恢复结构化信息。

手写文字：识别文档中的手写文字、印刷文字信息，并将识别的结构化结果以 JSON 格式返回给用户。

文字识别服务通常可以通过以下两种方式进行调用，一种是调用华为云提供的 SDK，另一种是调用相应服务的 API 接口。用户首先需要在华为云“EI 企业智能>人工智能>文字识别 OCR”页面开通相关服务（服务只需要开通一次即可，后面使用时无需再申请）。其次，利用 API 接口的开发者可以在华为云提供的 [API Explorer](#) 对不同服务的接口进行调测，同时 API Explorer 上提供有多种编程语言的 SDK 代码示例可供开发者参考使用。

3.2 实验目的

本实验主要介绍了使用华为云文字识别服务，通过本实验学员将了解如何利用华为云的文字识别服务进行通用文字识别、通用表格识别、手写文字识别功能。目前华为云有提供基于 Python 语言的文字识别 SDK，本实验将指导学员理解和掌握如何使用 Python 进行通用文字识别、表格识别、手写文字识别业务的开发方法和技巧。

3.3 实验准备

3.3.1 环境准备

1. 注册并登录华为云管理控制台。
-

2. 了解文字识别相关文档，详见 https://support.huaweicloud.com/api-ocr/ocr_03_0047.html。
3. 开通文字识别服务：登录文字识别管理控制台（<https://console.huaweicloud.com/ocr/?region=cn-north-4>），依次选择左侧的“总览”，“通用文字识别”、“通用表格识别”和“手写文字识别”，分别在界面单击“开通服务”。服务开通一次即可，后续使用时无需再开通。相关服务资费详情请参考华为云价格计算器。
4. 准备华为云账号的AK/SK。如果之前可以获取过，可以继续使用之前的AK/SK。如果之前没有生成过AK/SK，可登录华为云，在用户名处点击“我的凭证”，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。之后的实验不用再新增，可以直接使用此AK/SK信息。
5. 准备project_id。如果之前已经获取过，还可以继续使用之前的project_id。如果没有获取过，可在“我的凭证”界面的项目列表中查看项目ID，复制需要使用区域的项目ID为自己的project_id。

项目ID	项目	所属区域
07693c7eba00fe22f9	cn-north-4	华北-北京四
07691909f080266a2f4	cn-east-2	华东-上海二

6. 已经安装好 Python 环境，Python SDK 适用于 Python3。

3.3.2 SDK 安装

请确认已安装 Python 包管理工具 pip，并确认安装好 setuptools，requests 和 websocket-client，在 Jupyter Notebook 界面可通过“!pip list”（在 Jupyter Notebook 的 ipynb 记事本 Cell 中键入“!”表示调用系统命令）查看已安装包列表。如果没有安装，可以在系统终端中激活 Jupyter Notebook 所使用的虚拟环境并执行以下命令安装：

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

3.4 实验步骤

该实验需要安装华为公有云服务提供的文字识别 SDK，通过调用 SDK 底层接口服务并通过 AK/SK 信息进行用户身份认证。本实验就是通过 SDK 来调用文字识别的服务，并在 Jupyter Notebook 中实验，具体步骤如下：

步骤 1 导入所需要的包

```
# coding: utf-8

from huaweicloudsdkcore.auth.credentials import BasicCredentials
```

```
from huaweicloudsdkocr.v1.region.ocr_region import OcrRegion
from huaweicloudsdkcore.exceptions import exceptions
from huaweicloudsdkocr.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
credentials = BasicCredentials(ak, sk) \

client = OcrClient.new_builder() \
    .with_credentials(credentials) \
    .with_region(OcrRegion.value_of("cn-north-4")) \
    .build()
```

步骤 3 执行通用文字识别任务

GeneralTextRequestBody 函数 body 部分可以填入 image、url、detect_direction 等参数。

image: 图像数据，base64 编码，要求 base64 编码后大小不超过 10MB。图片最小边不小于 15px，最长边不超过 4096px。支持 JPEG、JPG、PNG、BMP、TIFF 格式。

url: 此项与 image 参数二选一，支持公网 HTTP/HTTPS URL，华为云 OBS 的 URL。

detect_direction: 图片朝向检测开关，True 为检测图片朝向，False 为不检测，默认值为 False。

quick_mode: 快速模式开关，针对单行文字图片（要求图片只包含一行文字，且文字区域占比超过 50%），打开时可以更快返回识别结果。True 为打开快速模式，False 为关闭快速模式，默认值为 False。

character_mode: 单字符模式开关。True 为打开单字符模式，False 为关闭单字符模式，默认值为 False，即不返回单个文本行的单字符信息。

```
try:
    request = RecognizeGeneralTextRequest()
    request.body = GeneralTextRequestBody(
        url="https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-
        AI/V3.5/chapter5/zh-cn_image_0288038182.png "
    )
    response = client.recognize_general_text(request)
    print(response)
except exceptions.ClientRequestException as e:
    print(e.status_code)
    print(e.request_id)
    print(e.error_code)
    print(e.error_msg)
```

输出结果：

direction: 图片朝向。当 **detect_direction** 为 false 时，该字段为-1。当 **detect_direction** 为 true 时，该字段为图片逆时针旋转角度，值区间为 0~359。

words_block_count: 检测到的文字块数目。

words_block_list: 识别文字块列表。输出顺序从左到右，先上后下。

```
{
  "result": {
    "direction": -1.0,
    "words_block_count": 10,
    "words_block_list": [
      {
        "words": "撤，还是不撤？",
        "location": [[161, 128], [885, 122], [886, 247], [162, 253]],
        "confidence": 0.9438
      },
      {
        "words": "让我更骄傲的是公司在大灾面前的表现。",
        "location": [[334, 460], [1934, 465], [1933, 583], [333, 578]],
        "confidence": 0.9992
      },
      {
        "words": "2011 年 3 月 11 日 14 时 46 分，日本东北部海域发生里氏 9.0 级",
        "location": [[326, 632], [2848, 626], [2848, 750], [326, 756]],
        "confidence": 0.9855
      },
      {
        "words": "地震并引发海啸。那一刻，我们正在距离东京 100 公里的热海开会，",
        "location": [[147, 804], [2845, 793], [2846, 924], [148, 935]],
        "confidence": 0.9965
      },
      {
        "words": "感觉“咚”地被震了一下。面对地震，大家都很镇定，直到看到电",
        "location": [[145, 981], [2859, 959], [2860, 1092], [146, 1113]],
        "confidence": 0.9999
      },
      {
        "words": "视上触目惊心的画面：15 时 25 分，海啸到达陆前高田市海岸；15 时",
        "location": [[142, 1157], [2862, 1135], [2863, 1267], [143, 1289]],
        "confidence": 0.9977
      },
      {
        "words": "26 分，海啸到达陆前高田市中心；15 时 43 分，陆前高田市依稀只能",
        "location": [[142, 1334], [2871, 1313], [2872, 1445], [143, 1466]],
        "confidence": 0.9991
      },
      {
        "words": "看到四层高的市府大楼的屋顶，一瞬间，城镇就变成了汪洋.....对",
        "location": [[140, 1514], [2871, 1492], [2872, 1625], [141, 1646]],
        "confidence": 0.9997
      },
      {
        "words": "我来说，地震跟家常便饭一样，可眼前的灾难比以往任何一次都要",
        "location": [[142, 1694], [2871, 1673], [2872, 1806], [143, 1827]],
        "confidence": 0.9998
      },
      {
        "words": "惨烈，完全超出了我的预期。",
        "location": [[145, 1878], [1314, 1875], [1315, 1993], [145, 1996]],
        "confidence": 0.9999
      }
    ]
  }
}
```

步骤 4 执行通用表格识别任务

GeneralTableRequestBody 函数 body 部分可以填入 image、url、return_text_location 等参数。

image: 图像数据，base64 编码，要求图片最小边不小于 15px，最长边不超过 8192px，支持 JPEG、JPG、PNG、BMP、TIFF 格式。

url: 此项与 image 参数二选一，支持公网 HTTP/HTTPS URL，华为云 OBS 的 URL。

return_text_location: 返回文本块坐标及单元格坐标信息，True 为返回文本块坐标及单元格坐标信息，False 为不返回，默认值为 False。

return_confidence: 返回置信度开关，True 为返回置信度，False 为不返回，默认值为 False。

return_excel: 返回表格转换 Microsoft Excel 的 base64 编码字段。True 为返回'excel'字段,表示 xlsx 格式的表格识别结果的 base64 编码，False 为不返回，默认值为 False。

```
try:
    request = RecognizeGeneralTableRequest()
    request.body = GeneralTableRequestBody(
        url="https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-
        AI/V3.5/chapter5/zh-cn_image_0282767866.png"
    )
    response = client.recognize_general_table(request)
    print(response)
except exceptions.ClientRequestException as e:
    print(e.status_code)
    print(e.request_id)
```



```
print(e.error_code)
print(e.error_msg)
```

输出结果：

words_region_count：文字区域数目。

words_region_list：文字区域识别结果列表，输出顺序从左到右，先上后下。

excel：表格图像转换为 excel 的 Base64 编码，图像中的文字和表格按位置写入 excel。对返回的 excel 编码可用 base64.b64decode 解码并保存为.xlsx 文件。

```
{
  "result": {
    "words_region_count": 2,
    "words_region_list": [
      {
        "type": "table",
        "words_block_count": 23,
        "words_block_list": [
          {
            "words": "门诊检验报告单",
            "rows": [0],
            "columns": [0, 1, 2, 3]
          },
          {
            "words": "***血常规 (5 分类)",
            "rows": [1],
            "columns": [0, 1, 2, 3]
          },
          {
            "words": "标本状态: 正常      临床诊断: 1.慢性扁桃体炎",
            "rows": [2],
            "columns": [0, 1, 2, 3]
          },
          {
            "words": "检验项目",
            "rows": [3],
            "columns": [0]
          },
          {
            "words": "结果",
            "rows": [3],
            "columns": [1]
          },
          {
            "words": "参考范围",
            "rows": [3],
            "columns": [2]
          },
          {
            "words": "单位",
            "rows": [3],
            "columns": [3]
          },
          {
            "words": "中性细胞百分率 (NEL%)",
            "rows": [4],
            "columns": [0]
          },
          {
            "words": "77.1",
            "rows": [4],
            "columns": [1]
          },
          {
            "words": "40-75",
            "rows": [4],
            "columns": [2]
          },
          {
            "words": "%",
            "rows": [4],
            "columns": [3]
          },
          {
            "words": "淋巴细胞百分率 (LYM%)",
            "rows": [5],
            "columns": [0]
          },
          {
            "words": "8.8",
            "rows": [5],
            "columns": [1]
          },
          {
            "words": "20-50",
            "rows": [5],
            "columns": [2]
          },
          {
            "words": "%",
            "rows": [5],
            "columns": [3]
          },
          {
            "words": "单核细胞百分率 (MONO%)",
            "rows": [6],
            "columns": [0]
          },
          {
            "words": "7.1",
            "rows": [6],
            "columns": [1]
          },
          {
            "words": "3.0-10.0",
            "rows": [6],
            "columns": [2]
          },
          {
            "words": "%",
            "rows": [6],
            "columns": [3]
          },
          {
            "words": "红细胞计数 (RBC)",
            "rows": [7],
            "columns": [0]
          },
          {
            "words": "6.66",
            "rows": [7],
            "columns": [1]
          },
          {
            "words": "4.3-5.8",
            "rows": [7],
            "columns": [2]
          },
          {
            "words": "%",
            "rows": [7],
            "columns": [3]
          }
        ],
        "type": "text",
        "words_block_count": 3,
        "words_block_list": [
          {
            "words": "送检医生:"
          },
          {
            "words": "检验者:"
          },
          {
            "words": "审核者:"
          }
        ]
      }
    ]
  }
}
```

步骤 5 执行手写文字识别任务

HandwritingRequestBody 函数 body 部分可以填入 image、url、quick_mode 等参数。

image：图像数据，base64 编码，要求 base64 编码后大小不超过 10MB。图片最小边不小于 8px，最长边不超过 8192px，支持 JPEG、JPG、PNG、BMP、TIFF 格式。

url：此项与 image 参数二选一，支持公网 HTTP/HTTPS URL，华为云 OBS 的 URL。

quick_mode：快速模式开关，针对单行文字图片（要求图片只包含一行文字，且文字区域占比超过 50%），打开时可以更快返回识别结果。True 为打开快速模式，False 为关闭快速模式，默认值为 False。

detect_direction：图片朝向检测开关，True 为检测图片朝向，False 为不检测，默认值为 False。

char_set：字符集设置,用户可以根据实际需要限定输出字符集范围。可选“digit”：数字模式，“letter”：大小写字母模式，“digit_letter”：数字+字母模式，“general”：数字+字母+中文模式。

```
try:
    request = RecognizeHandwritingRequest()
    request.body = HandwritingRequestBody(
        url="https://certification-data.obs.cn-north-4.myhuaweicloud.com/CHS/HCIA-AI/V3.5/chapter5/zh-cn_image_0288038984.png"
    )
    response = client.recognize_handwriting(request)
```

```
print(response)
except exceptions.ClientRequestException as e:
    print(e.status_code)
    print(e.request_id)
    print(e.error_code)
    print(e.error_msg)
```

输出结果：

`words_block_count`：检测到的文字块数目。

`words_block_list`：识别文字块列表。输出顺序从左到右，从上到下。

```
{"result": {"words_block_count": 2, "words_block_list": [{"words": "明月几时有", "type": "text",
"confidence": 0.9841, "location": [[357, 152], [885, 152], [885, 266], [357, 266]]}, {"words": "把酒问青天",
"confidence": 0.9873, "location": [[320, 328], [894, 306], [899, 429], [325, 452]]}]}}
```

3.5 思考题

小明想自己手写一份约两千字材料并利用华为云接口进行识别检测，你会推荐他使用哪个接口？并尝试帮助小明编写一段从本地上传图片的代码。

3.6 实验小结

本章主要介绍了应用华为公有云上的图像标签检测服务的具体操作，主要是通过配置用户的认证信息及准备好所需检测材料信息并调用 SDK 进行相关检测服务。本章中主要针对用户 AK/SK 认证的方式进行了系统的介绍和说明，为使用通用文字识别、通用表格识别、手写文字识别服务提供了实际的操作指导。

4 语音合成和语音识别

4.1 实验简介

华为云上的语音交互服务中，有语音合成和语音识别服务，本实验的内容是定制版语音合成和定制版的一句话识别服务。

语音合成（Text To Speech, TTS），又称文语转换，是一种将文本转换成逼真语音的服务。语音合成以开放 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 获取语音合成结果，将用户输入的文字合成为音频。通过音色选择、自定义音量、语速，为企业和个人提供个性化的发音服务。

语音识别（Automatic Speech Recognition, ASR），将口述音频转换为文本。语音识别以开放 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 获取语音识别结果。当前语音识别提供了短语音识别和长语音识别功能，短语音识别对时长较短的语音识别速度更快，长语音识别对时长较长的录音文件转写效果更好。

一句话识别服务：可以实现 1 分钟以内、不超过 4MB 的音频到文字的转换。对于用户上传的完整的录音文件，系统通过处理，生成语音对应文字内容。

语音交互服务通常可以通过以下两种方式进行调用，一种是调用华为云提供的 SDK，另一种是调用相应服务的 API 接口。用户可利用 API 接口的开发者可以在华为云提供的 [API Explorer](#) 对不同服务的接口进行调测，同时 API Explorer 上提供有多种编程语言的 SDK 代码示例可供开发者参考使用。

4.2 实验目的

本实验主要介绍了使用华为云语音交互服务，通过本实验学员将了解如何利用华为云的语音交互服务进行语音合成、语音识别功能。目前华为云有提供基于 Python 语言的语音交互 SDK，本实验将指导学员理解和掌握如何使用 Python 进行语音合成、语音识别的开发方法和技巧。

4.3 实验准备

4.3.1 环境准备

1. 注册并登录华为云管理控制台。
-

2. 了解语音合成与语音识别相关文档，详见 https://support.huaweicloud.com/api-sis/sis_03_0111.html 和 https://support.huaweicloud.com/api-sis/sis_03_0040.html。
3. 准备华为云账号的AK/SK。如果之前可以获取过，可以继续使用之前的AK/SK。如果之前没有生成过AK/SK，可登录华为云，在用户名处点击“我的凭证”，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。之后的实验不用再新增，可以直接使用此AK/SK信息。
4. 准备project_id。如果之前已经获取过，还可以继续使用之前的project_id。如果没有获取过，可在“我的凭证”界面的项目列表中查看项目ID，复制需要使用区域的项目ID为自己的project_id。

项目ID	项目	所属区域
07693c7eba000fe22f9	cn-north-4	华北-北京四
07691909f080266a2f4	cn-east-2	华东-上海二

5. 已经安装好 Python 环境，Python SDK 适用于 Python3。

4.3.2 SDK 获取和配置

1. 下载语音交互服务的 Python SDK (<https://mirrors.huaweicloud.com/sis-sdk/python/huaweicloud-python-sdk-sis-1.0.0.rar>) 并解压。data 文件夹中的数据我们可以用，代码与 data 文件夹同级别即可，我们也可以使用自己的数据放在 data 文件夹中。
2. 请确认已安装 Python 包管理工具 setuptools，请确认已安装 requests 和 websocket-client，可通过“pip list”命令查看已安装列表。如果没有安装，请使用以下命令安装：

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

4.4 实验步骤

该实验需要在华为公有云服务上下载语音交互服务的 SDK，通过 AK/SK 信息进行身份认证并调用 SDK 底层接口服务进行 Restful 服务请求的提交。本实验是通过 SDK 来调用语音合成和语音识别服务的，并在 Jupyter Notebook 中进行实验。以下的实验我们先进行语音合成生成语音数据，然后再用语音识别对语音数据进行识别。具体步骤如下：

4.4.1 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。通过音色选择、自定义音量、语速，为企业和个人提供个性化的发音服务。

步骤 1 导入所需要的包

```
# -*- coding: utf-8 -*-
from huaweicloud_sis.client.tts_client import TtsCustomizationClient
```

```
from huaweicloud_sis.bean.tts_request import TtsCustomRequest
from huaweicloud_sis.bean.sis_config import SisConfig
from huaweicloud_sis.exception.exceptions import ClientException
from huaweicloud_sis.exception.exceptions import ServerException
import json
```

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
region = "cn-north-4" #默认使用北京-4 区，对应的区域代码即为 cn-north-4
```

步骤 3 配置数据和保存路径

```
text = '明天有雨吗？需要带伞吗?' # 待合成文本，不超过 500 字
path = 'test.wav' # 保存路径
```

步骤 4 初始化客户端

```
config = SisConfig()
config.set_connect_timeout(5) # 设置连接超时，单位 s
config.set_read_timeout(10) # 设置读取超时，单位 s
ttsc_client = TtsCustomizationClient(ak, sk, region, project_id, sis_config=config)
```

步骤 5 构造请求

```
ttsc_request = TtsCustomRequest(text)
# 设置请求，所有参数均可不设置，使用默认参数
# 设置属性字符串， language_speaker_domain, 默认 chinese_xiaoyan_common, 参考 api 文档
# 不同发音人参考 https://support.huaweicloud.com/api-sis/sis\_03\_0111.html 中表 5、表 6
ttsc_request.set_property('chinese_xiaoyan_common')
# 设置音频格式，默认 wav，可选 mp3 和 pcm
ttsc_request.set_audio_format('wav')
# 设置采样率，8000 or 16000, 默认 8000
ttsc_request.set_sample_rate('8000')
# 设置音量，[0, 100]，默认 50
ttsc_request.set_volume(50)
# 设置音高，[-500, 500]，默认 0
ttsc_request.set_pitch(0)
# 设置音速，[-500, 500]，默认 0
ttsc_request.set_speed(0)
# 设置是否保存，默认 False
ttsc_request.set_saved(True)
# 设置保存路径，只有设置保存，此参数才生效
ttsc_request.set_saved_path(path)
```

步骤 6 语音合成测试

#发送请求，返回结果。如果设置保存，可在指定路径里查看保存的音频。

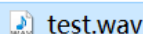
```
result = ttsc_client.get_ttsc_response(ttsc_request)
print(json.dumps(result, indent=2, ensure_ascii=False))
```

输出结果：

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

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。data 代表语音数据，base64 编码格式返回。

保存的语音数据如下：



4.4.2 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 -*-
from huaweicloud_sis.client.asr_client import AsrCustomizationClient
from huaweicloud_sis.bean.asr_request import AsrCustomShortRequest
from huaweicloud_sis.bean.asr_request import AsrCustomLongRequest
from huaweicloud_sis.exception.exceptions import ClientException
from huaweicloud_sis.exception.exceptions import ServerException
from huaweicloud_sis.utils import io_utils
from huaweicloud_sis.bean.sis_config import SisConfig
import json
```

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
region = "cn-north-4" #默认使用北京-4 区，对应的区域代码即为 cn-north-4
```

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_8k_common' # language_sampleRate_domain, 如 chinese_8k_common, 详见 api 文档
```

步骤 4 初始化客户端

```
config = SisConfig()
config.set_connect_timeout(5) # 设置连接超时
config.set_read_timeout(10) # 设置读取超时
asr_client = AsrCustomizationClient(ak, sk, region, project_id, sis_config=config)#初始化客户端
```

步骤 5 构造请求

```
data = io_utils.encode_file(path)
asr_request = AsrCustomShortRequest(path_audio_format, path_property, data)
# 所有参数均可不设置, 使用默认值
# 设置是否添加标点, yes or no, 默认 no
asr_request.set_add_punc('yes')
```

步骤 6 语音识别测试

```
#发送请求, 返回结果,返回结果为 json 格式
result = asr_client.get_short_response(asr_request)
print(json.dumps(result, indent=2, ensure_ascii=False))
```

输出结果:

```
{
  "trace_id": "f88ed78f-9d05-41de-afb7-1f9e12f82dcc",
  "result": {
    "text": "明天有雨吗? 需要带伞吗? ",
    "score": 0.7158617532616159
  }
}
```

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)。

4.5 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 (语音合成和语音识别) 进行实验的具体操作, 主要是通过配置用户的认证信息及准备好所需检测材料信息并调用 SDK 进行相关检测服务。本章中主要针对用户 AK/SK 认证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。