**2022-2023学年秋季学期东北大学**

**《自然语言处理》项目实践报告**

**姓 名** 胡婧娴

**专 业**  计算机科学与技术

**学 号**  2201799

**学 院**  计算机科学与工程学院

**任课教师** 肖桐

**2023 年 1月 15 日**

**一、实验主题**

本次实验主要是实现基于卷积神经网络模型的水果图像分类识别问题。

卷积神经网络（VGG）是当前最流行的CNN模型，通过本次实验，完成对水果数据集的分类，以更好地理解卷积神经网络的结构模型。

**二、实验环境要求**

win10,pycharm

Python3.6

keras2.2.0

numpy1.16.0

pillow8.1.1

scipy1.5.4

tensorflow-gpu1.11.0

1. **实验设计**

整个实验总体上可分为以下5个步骤

1. 数据采集

2. 数据预处理

3. 搭建Tensorflow构建卷积神经网络模型

4. 模型训练

5. 查看模型在测试集上的准确率

**四、实验过程**

**1数据获取与数据集介绍**

数据来自于网上公开数据集（获取地址：<https://www.kaggle.com/moltean/fruits）>

公开水果数据集Fruits-360，该数据集中共有131类水果，每张图片的像素均为100x100

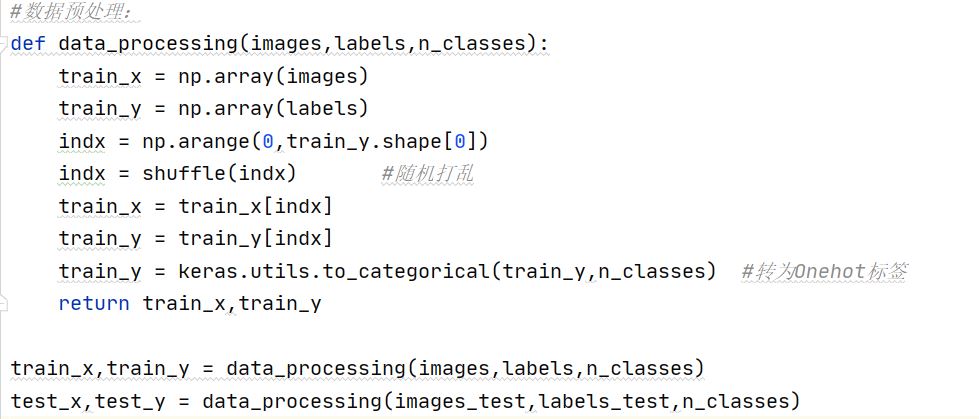
**2数据载入与数据处理**

原来的数据集有131种分类有些过多，所以为了节省训练时间，本次实验最后实际进行过程中只随机选了37类做实验（在资源中把131个种类全部给出了）。分别建立一个训练文件夹和一个测试文件夹，把想要训练的分类从原数据文件夹拷到里面即可。

读图函数方法很多，这里通过os.listdir()这个函数来实现文件夹下的循环读图，返回的数据有所有的图集数据以列表形式存放；其对应的标签，这里的标签由于其文件夹是字符串，不便于后期的独热编码转换，故而存放为外循环的迭代数

**

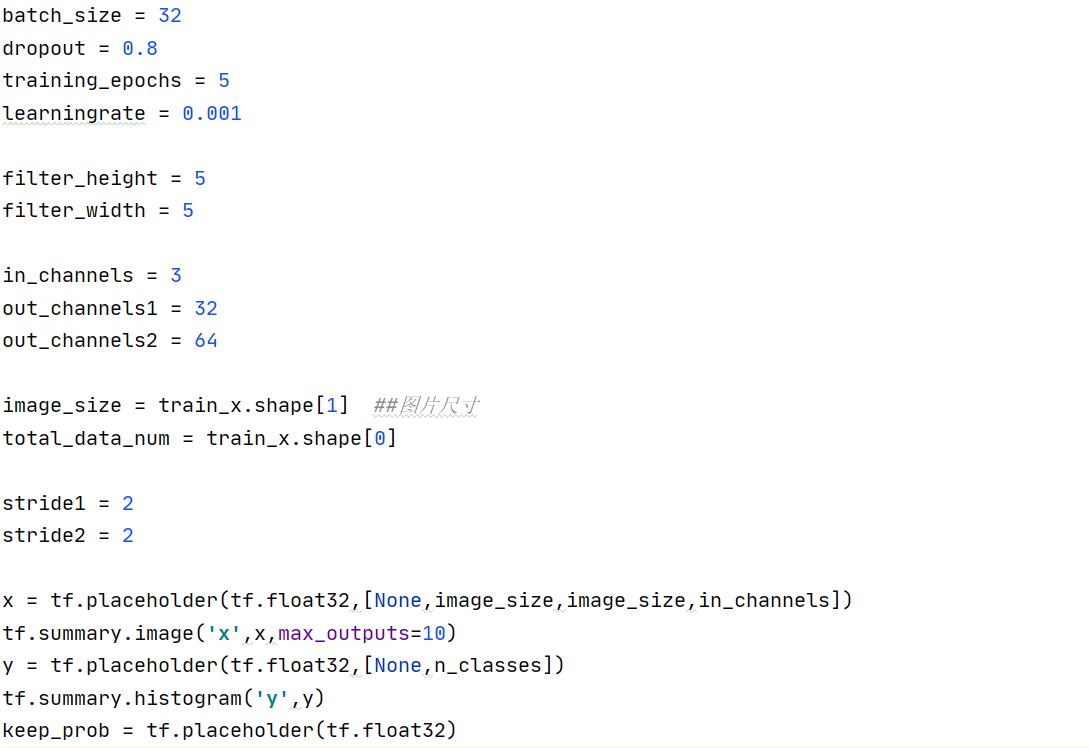
在图像预处理这个步骤中，查阅资料时发现有很多种处理方式，如图像的去噪、平移、反转、灰度化、裁剪等，但是有些效果并不理想，用处不大，因此在本次实验中并未用的这些上述的处理方式，而是简单的把图像数据集转换为数组形式，并将其乱序，对变迁数据集进行keras的独热编码。



**3搭建模型**

使用了经典的卷积神经网络LeNet-5模型，特别的本实验中，使用了全局平均池化层代替全连接层。

1. 神经网络参数配置具体如下



1. 卷积神经网络结构搭建



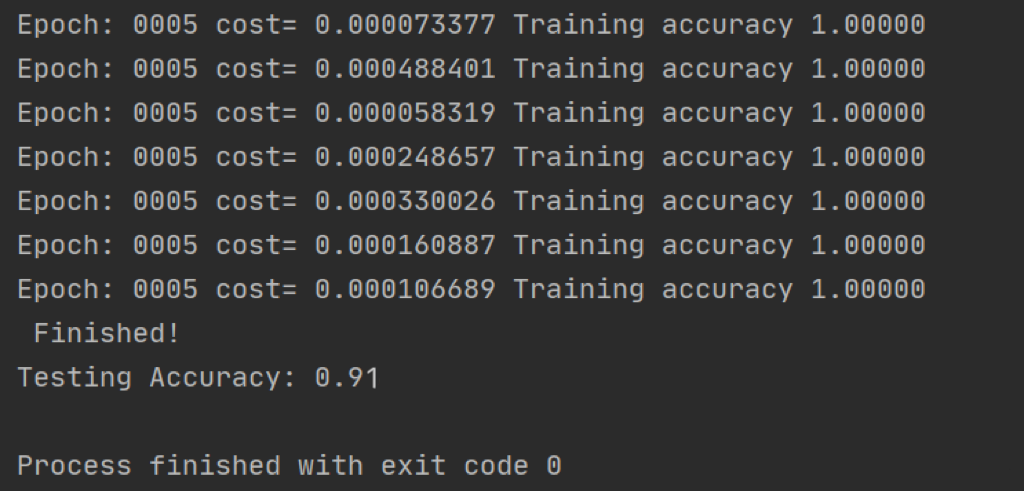


**4训练数据并查看模型在测试集上的准确率**



**5运行结果**

测试了37类水果，准确率可以达到0.91左右



**五、实验小结**

通过本次实验，对于卷积神经网络的搭建有了更多的认识，并且通过水果分类问题这个实例，了解了解决该类图像处理问题的具体步骤和思路。

需要改进的地方还有很多，比如试验结束后在网上查看了一些资料，很多人都在该问题上使用了，一些博主表示在相同训练次数以及相同学习率的情况下，Adam的准确率更高一点，因此在他们的实现过程中还有一个优化器的过程（如下图所示）。此外还有博主提到，对于超参数学习率来说，该模型采用的是固定常数的学习率，本实验中是0.001，也可以使用具有线性变化的学习率进行训练，有可能会获得更好的模型精度。以上这两个问题都是我还需要尝试和改进的地方。

optimizer=tf.train.AdamOptimizer(0.0009).minimize(cross\_entropy) *##选择优化器以及学习率*

**六、附录**

源码：

import os

from skimage import io,transform

import numpy as np

from sklearn.utils import shuffle

import keras

import tensorflow as tf

#如果对所有种类进行训练和测试，选择全部Training和Test

dir\_path1 = 'fruits-360/Training1'

dir\_path2 = 'fruits-360/Test1'

#读取数据

def load\_data(dir\_path):

images = []

labels = []

file = os.listdir(dir\_path) #返回一个列表：包含dir\_path里的文件名，即分类

n\_classes = len(file) #分类的数量

n = 0

for l in file:

img = os.listdir(dir\_path+'/'+ l)

for i in img:

img\_path = dir\_path+'/'+ l+'/'+ i

labels.append(int(n))

images.append(io.imread(img\_path))

n += 1

return images,labels,n\_classes

images,labels,n\_classes = load\_data(dir\_path1)

images\_test,labels\_test,\_ = load\_data(dir\_path2)

print("种类：",n\_classes)

#数据预处理：

def data\_processing(images,labels,n\_classes):

train\_x = np.array(images)

train\_y = np.array(labels)

indx = np.arange(0,train\_y.shape[0])

indx = shuffle(indx) #随机打乱

train\_x = train\_x[indx]

train\_y = train\_y[indx]

train\_y = keras.utils.to\_categorical(train\_y,n\_classes) #转为Onehot标签

return train\_x,train\_y

train\_x,train\_y = data\_processing(images,labels,n\_classes)

test\_x,test\_y = data\_processing(images\_test,labels\_test,n\_classes)

#神经网络参数配置

batch\_size = 32

dropout = 0.8

training\_epochs = 5

learningrate = 0.001

filter\_height = 5

filter\_width = 5

in\_channels = 3

out\_channels1 = 32

out\_channels2 = 64

image\_size = train\_x.shape[1] #图片尺寸

total\_data\_num = train\_x.shape[0]

stride1 = 2

stride2 = 2

x = tf.placeholder(tf.float32,[None,image\_size,image\_size,in\_channels])

y = tf.placeholder(tf.float32,[None,n\_classes])

keep\_prob = tf.placeholder(tf.float32)

Weights = {"conv1":tf.Variable(tf.random\_normal([filter\_height,filter\_width,in\_channels,out\_channels1])),

"conv2":tf.Variable(tf.random\_normal([filter\_height,filter\_width,out\_channels1,out\_channels2])),

"conv3":tf.Variable(tf.random\_normal([filter\_height,filter\_width,out\_channels2,n\_classes])),}

bias = {"conv1":tf.Variable(tf.random\_normal([out\_channels1])),

"conv2":tf.Variable(tf.random\_normal([out\_channels2])),

"conv3":tf.Variable(tf.random\_normal([n\_classes]))}

#定义卷积层的生成函数

def conv2d(x, W, b, stride=1):

x = tf.nn.conv2d(x, W, strides=[1, stride, stride, 1], padding="SAME")

x = tf.nn.bias\_add(x, b)

return tf.nn.relu(x)

#定义池化层的生成函数

def maxpool2d(x, stride=2):

return tf.nn.max\_pool(x, ksize=[1, stride, stride, 1], strides=[1, stride, stride, 1], padding="SAME")

#定义卷积神经网络生成函数

def conv\_net(x, Weights, bias, dropout):

#卷积层1

conv1 = conv2d(x, Weights['conv1'], bias['conv1']) ##100\*100\*32

conv1 = maxpool2d(conv1, stride1) ##经过池化层1 shape：50\*50\*32

#卷积层2

conv2 = conv2d(conv1, Weights['conv2'], bias['conv2']) ##50\*50\*64

conv2 = maxpool2d(conv2, stride2) ##经过池化层2 shape:25\*25\*64

#Dropout层防止预测数据过拟合

conv2 = tf.nn.dropout(conv2, dropout)

#全局平均池化层

conv3 = conv2d(conv2, Weights['conv3'], bias['conv3']) ##25\*25\*10

prediction = tf.nn.avg\_pool(conv3,ksize=[1,25,25,1],strides=[1,25,25,1],padding='SAME')

return tf.reshape(prediction,[-1,n\_classes])

#优化预测准确率

prediction = conv\_net(x, Weights, bias, keep\_prob) #生成卷积神经网络

loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits=prediction, labels=y)) #交叉熵损失函数

optimizer = tf.train.AdamOptimizer(learningrate).minimize(loss) #优化器

correct\_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))

#初始会话并开始训练过程

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

#启动循环开始训练

for epoch in range(training\_epochs):

total\_batch = int(total\_data\_num / batch\_size)

#遍历全部数据集

for i in range(total\_batch):

batch\_x, batch\_y = train\_x[i\*batch\_size:batch\_size \* (i+1),:],train\_y[i\*batch\_size:batch\_size \* (i+1),:]

\_, acc,Loss = sess.run([optimizer,accuracy,loss], feed\_dict={x: batch\_x,y: batch\_y,keep\_prob:dropout})

# Compute average loss

print("Epoch:", '%04d' % (epoch + 1), "cost=", "{:.9f}".format(Loss), "Training accuracy", "{:.5f}".format(acc))

print(" Finished!")

#测试模型

test\_feed = {x: test\_x[0:100],y:test\_y[0:100],keep\_prob:1}

print('Testing Accuracy:', sess.run(accuracy, feed\_dict=test\_feed))