使用LeNet-5框架对CIFAR-10数据集进行分类

曾祥益 18051936 联系方式：QQ:2950244886

**网络结构如下：**

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Parameters #

=============================================

conv2d (Conv2D) (None, 28, 28, 6) 456

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d (MaxPooling2D) (None, 14, 14, 6) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_1 (Conv2D) (None, 10, 10, 16) 2416

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2D) (None, 5, 5, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 400) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 120) 48120

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 84) 10164

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 10) 850

===========================================

Total parameters: 62,006

Trainable parameters: 62,006

Non-trainable parameters: 0

首先由2层卷积池化层组成：第一层由6个5\*5的卷积核组成,池化部分采用最大池化；第二层由16个5\*5的卷积核组成,之后采用最大池化.后面加上3层全连接层,分别为400🡪120、120🡪84、84🡪10.

**数据导入及处理**：

CIFAR-10数据集已经将总共的数据分为6部分——5个train\_batch和一个test\_batch.数据的读取函数在CIFAR\_10数据集官网已经给出,每个batch中有着10000个数据,总共60000条.每个batch中的读出的数据为dict类型,其中的两个词条——labels、data分别对应标签和对应的图像.本次代码中只取了这两个数据作处理.对data数据归一化,对labels数据one-hot化,之后进行训练.值得注意的是,这次的data数据与以前不同,这次是RGB图像,格式为(,32,32,3),但是data中读出来的格式为(,3072),其第二维度(3072)中每1024的数据,对应了一个颜色通道,因此需要对数据进行reshape,但是在尝试reshape之后,数据却是(,3,32,32),与目标的(,32,32,3)不符,查阅资料后,我了解到了一个函数可以任意调换数据的维度——transpose(),这样一来,就方便了很多,不用再单独对数据的维度进行考虑.训练数据集采用了数据集中前40000条数据,验证集采用了batch5中的10000条数据,最后测试集中10000条数据用来评估模型.

**网络设置及建立：**

因为采用了框架进行训练,网络写起来会显得很清晰,在layers中,按照LeNet-5的架构,先置入两层卷积池化层,再连上三层全连接,即可借助框架构建出网络结构,再通过tensorflow中的Sequential()函数建立实例,网络模型即可建立完成.

**网络训练：**

再对建立好的模型中的习率、损失计算方式、评估指标等参数进行修改之后,即可编译模型,最后,通过tensorflow.keras中的fit函数,传入待训练的数据,设置好迭代次数等参数,即可进行训练.下面是一些我对该函数参数的理解：

Epochs：所有的样本遍历一遍,相当于所有的数据都正向传播了一次,反向传播了一次.

Shuffle：是否打乱顺序,有助于防止过拟合,增强网络的泛化能力.

Batch\_size：每次通过该网络的数据量,网络每运行一次batch就更新一次参数,step\*batch\_size即为总共的数据量,一次epoch中含有多个step,即一次迭代中网络参数会更新step数量次.

**其他设置：**

设立了一个early\_stopping,如果迭代多轮,模型效果任然没有提升或者甚至倒退,则停止训练,输出记录的最好的参数

设立了checkpoint,每迭代5轮保存一次网络参数,因为训练时间会非常长,设立保存点十分有必要.keras中的fit函数支持断点训练,在训练网络意外或者手动终止后,可以通过读取保存的保存点来载入网络参数,设置好初始迭代次数后,即可继续训练.

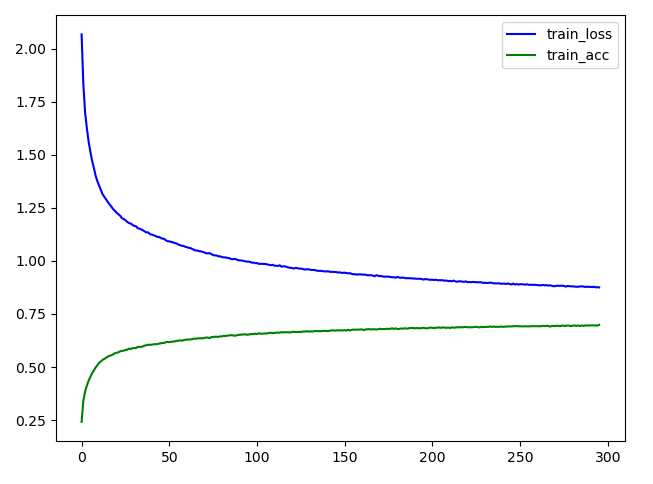
最后,保存网络机构后,该网络即可作为测试或者应用,通过框架中的评估模型,即可给出得分.

**训练结果：**

测试集正确率：63.95%



Loss曲线：



整体代码如下: import pickle

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers, Sequential

def unpickle(file):

with open(file, 'rb') as fo:

dict = pickle.load(fo, encoding='bytes')

return dict

def files(batch):

path =’E:\创新实践操作\cifar-10-python\cifar-10-batches-py' + str(batch)

dict = unpickle(path)

return dict

def Data(n):

file = files(n)

label = file[b'labels']

data = file[b'data']

ones = np.zeros((10000, 10))

for i, row in enumerate(ones):

row[label[i]] = 1 # 将标签one-hot化

data = data.reshape(10000, 3, 32, 32).transpose(0, 2, 3, 1) / 255 # 数据由10000,3,32,32 转换为10000,32,32,3

return data, ones

x\_train = []

y\_train = []

for i in range(1, 6):

x, y = Data(i)

x\_train.extend(x)

y\_train.extend(y) #载入batch1-4的数据

x\_train = np.array(x\_train)

y\_train = np.array(y\_train)

x\_val, y\_val = Data('test') #batch-test设置为验证集

x\_val = np.array(x\_val)

y\_val = np.array(y\_val)

# plt.imshow(x\_train[20][:, :, :]) #展示图片

# plt.show()

d\_shape = x\_train.shape[1:]

cnn\_layers = [

# unit1

layers.Conv2D(6, kernel\_size=[5, 5], strides=1, padding='valid', activation=tf.nn.sigmoid, input\_shape=d\_shape),

layers.MaxPool2D(pool\_size=[2, 2], padding='same', strides=2),

# 卷积池化

# unit2

layers.Conv2D(16, kernel\_size=[5, 5], strides=1, padding='valid', activation=tf.nn.sigmoid),

layers.MaxPool2D(pool\_size=[2, 2], padding='same', strides=2),

# 卷积池化

layers.Flatten(),

layers.Dense(120, ),

layers.Dense(84, ),

layers.Dense(10, activation=tf.nn.softmax),

# 三层全连接,最后用softmax激活输出

] # 网络结构

cnn = Sequential(cnn\_layers)

# 创建网络模型

cnn.summary()

cnn.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=1e-3), loss='categorical\_crossentropy',

metrics=['categorical\_accuracy']) # 编译模型,设置学习率、损失计算方式、评估指标

early\_stopping\_checkpoint = tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=30, restore\_best\_weights=True,

verbose=2,

mode='min') # 提早停止epoch(如果loss不再下降超过patient指定的轮次)

model\_checkpoint\_callback = keras.callbacks.ModelCheckpoint(

'training\_checkpoints/weights.{epoch:02d}-{val\_loss:.2f}.hdf5', period=5) # 检查点设置

# cnn.load\_weights('training\_checkpoints\weights.645-1.07.hdf5') # 载入检查点(载入参数权重)

history = cnn.fit(x=x\_train, y=y\_train, epochs=1000, verbose=2,

callbacks=[early\_stopping\_checkpoint, model\_checkpoint\_callback],

validation\_data=(x\_val, y\_val)) # 训练模型

np.savetxt('train\_loss.txt', history.history['loss'])

np.savetxt('train\_acc.txt', history.history['categorical\_accuracy'])

np.savetxt('valid\_loss.txt', history.history['val\_loss'])

np.savetxt('valid\_acc.txt', history.history['val\_categorical\_accuracy'])

# model = keras.models.load\_model('cnn.h5') #加载模型

# x\_t, y\_t = Data('test') #修改了’test\_batch‘名称,方便用Data函数导入

# x\_t = np.array(x\_t)

# y\_t = np.array(y\_t)

# model.evaluate(x\_t, y\_t,) #评估模型

cnn.save('cnn.h5') # 保存模型