1. VGG模型

前面已经介绍过了VGG网络模型，一共13层，这里说的13层指的是10层卷积层和3层全连接层，并没有包括池化层。下面代码详细地将VGG的13层网络模型复现，并用CIFAR100数据集进行训练，测试。

代码中附有详细的注释，从数据的预处理，到训练，再到测试。

1. 代码实现

import tensorflow as tf  
from tensorflow.keras import layers, optimizers, datasets, Sequential  
import os  
  
os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'  
tf.random.set\_seed(2345)  
  
***#VGG13总共13层，指的是10层卷积层和3层全连接层***"""  
一共5个单元的卷积层和池化层，每个单元2个卷积层和一个池化层  
"""  
  
***#卷积层和池化层***conv\_layers = [  
  
 ***#第一个单元 2个卷积和一个池化*** layers.Conv2D(64, kernel\_size=[3, 3], padding="same", activation=tf.nn.relu),  
 layers.Conv2D(64, kernel\_size=[3, 3], padding="same", activation=tf.nn.relu),  
 layers.MaxPool2D(pool\_size=[2, 2], strides=2, padding="same"),  
  
 ***#第二个单元*** layers.Conv2D(128, kernel\_size=[3, 3], padding="same", activation=tf.nn.relu),  
 layers.Conv2D(128, kernel\_size=[3, 3], padding="same", activation=tf.nn.relu),  
 layers.MaxPool2D(pool\_size=[2, 2], strides=2, padding="same"),  
  
 ***#第三个单元*** layers.Conv2D(256, kernel\_size=[3, 3], padding="same", activation=tf.nn.relu),  
 layers.Conv2D(256, kernel\_size=[3, 3], padding="same", activation=tf.nn.relu),  
 layers.MaxPool2D(pool\_size=[2, 2], strides=2, padding="same"),  
  
 ***#第四个单元*** layers.Conv2D(512, kernel\_size=[3, 3], padding="same", activation=tf.nn.relu),  
 layers.Conv2D(512, kernel\_size=[3, 3], padding="same", activation=tf.nn.relu),  
 layers.MaxPool2D(pool\_size=[2, 2], strides=2, padding="same"),  
  
 ***#第五个单元*** layers.Conv2D(512, kernel\_size=[3, 3], padding="same", activation=tf.nn.relu),  
 layers.Conv2D(512, kernel\_size=[3, 3], padding="same", activation=tf.nn.relu),  
 layers.MaxPool2D(pool\_size=[2, 2], strides=2, padding="same"),  
  
]  
  
***#数据预处理***def preprocess(x, y):  
 ***#数据到0-1之间*** x = tf.cast(x, dtype=tf.float32) / 255.  
 y = tf.cast(y, dtype=tf.int32)  
 return x,y  
  
***#数据自动下载加载***(x, y), (x\_test, y\_test) = datasets.cifar100.load\_data()  
print(x.shape, y.shape, x\_test.shape, y\_test.shape)  
***#标签 (64, 1)，需要squeeze掉 1***y = tf.squeeze(y, axis=1)  
y\_test = tf.squeeze(y\_test, axis=1)  
  
***#数据的处理  
#训练数据的处理***train\_db = tf.data.Dataset.from\_tensor\_slices((x, y))  
train\_db = train\_db.shuffle(1000).map(preprocess).batch(64)  
***#测试数据的处理***test\_db = tf.data.Dataset.from\_tensor\_slices((x\_test, y\_test))  
test\_db = test\_db.map(preprocess).batch(64)  
  
***#取出一个sample查看 每个样本数据的尺寸，标签的尺寸，最大值最小值***sample = next(iter(train\_db))  
print("sample:", sample[0].shape, sample[1].shape,  
 tf.reduce\_min(sample[0]), tf.reduce\_max(sample[0]))  
  
  
  
  
def main():  
  
 ***#卷积层*** conv\_net = Sequential(conv\_layers)  
  
 ***#test卷积层和池化层  
 #conv\_net.build(input\_shape=[None, 32, 32, 3])  
 # x = tf.random.normal([4, 32, 32, 3])  
 # out = conv\_net(x)  
 # print(out.shape)  
  
 #全连接层*** fc\_net = Sequential([  
 layers.Dense(256, activation=tf.nn.relu),  
 layers.Dense(128, activation=tf.nn.relu),  
 layers.Dense(100, activation=None),  
 ])  
  
 conv\_net.build(input\_shape=[None, 32, 32, 3])  
 fc\_net.build(input\_shape=[None, 512])  
  
 ***#设置优化器，优化学习率*** optimizer = optimizers.Adam(lr=1e-4)  
  
 ***#每个层键的参数变量结合起来*** variables = conv\_net.trainable\_variables + fc\_net.trainable\_variables  
  
 ***#处理完数据后，开始训练*** for epoch in range(50):  
 for step, (x, y) in enumerate(train\_db):  
 ***#自动求解梯度*** with tf.GradientTape() as tape:  
 ***#卷积层和池化层  
 #[b, 32, 32, 3] -> [b, 1, 1, 512]*** out = conv\_net(x)  
 ***#flatten -> [b, 512]*** out = tf.reshape(out, [-1, 512])  
 ***#全连接层  
 #[b, 512] -> [b, 100]*** logits = fc\_net(out)  
  
 ***#走完卷积层，池化层，全连接层得到输出值  
 #求解Loss值,需要将y的尺寸补齐到logits的尺寸  
 #[b] -> [b, 100]*** y\_onehot = tf.one\_hot(y, depth=100)  
 loss = tf.losses.categorical\_crossentropy(y\_onehot, logits, from\_logits=True)  
 ***#计算Loss的均值，每个样本有一个loss均值*** loss = tf.reduce\_mean(loss)  
  
 ***#反向传播，对所有的层间参数进行求导*** grads = tape.gradient(loss, variables)  
 ***#更新梯度*** optimizer.apply\_gradients(zip(grads, variables))  
  
 ***#每100个样本打印一次结果*** if step%100 == 0:  
 print(epoch, step, "loss:", float(loss))  
  
 ***#测试模型*** total\_num = 0  
 total\_correct = 0  
 for x, y in test\_db:  
 out = conv\_net(x)  
 out = tf.reshape(out, [-1, 512])  
 logits = fc\_net(out)  
 ***#softmax转换成概率*** prob = tf.nn.softmax(logits, axis=1)  
 ***#选择概率最大的作为预测值*** pred = tf.argmax(prob, axis=1)  
 pred = tf.cast(pred, dtype=tf.int32)  
  
 ***#计算正确预测的数量*** correct = tf.cast(tf.equal(pred, y), dtype=tf.int32)  
 correct = tf.reduce\_sum(correct)  
  
 total\_num += x.shape[0]  
 total\_correct += int(correct)  
  
 ***#计算正确率*** acc = total\_correct / total\_num  
 print(epoch, "acc:", acc)  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()