测试用的仍然是LetNet实验中的数据集

import os

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

BATCH\_SIZE = 64

LR = 0.001

EPOCHS = 1

MAX\_STEPS = 100

mnist = input\_data.read\_data\_sets("MNIST", one\_hot=True)

class AlexNet(object):

def \_\_init\_\_(self, num\_classes, keep\_prob):

self.num\_classes = num\_classes

self.keep\_prob = keep\_prob

#模型

def create(self, X):

X = tf.reshape(X, [-1, 28, 28, 1])

conv\_layer1 = self.conv\_layer(X, 11, 96, 4, "Layer1")

pool\_layer1 = self.pool\_layer(conv\_layer1, 3, 2)

conv\_layer2 = self.conv\_layer(pool\_layer1, 5, 256, 2, "Layer2")

pool\_layer2 = self.pool\_layer(conv\_layer2, 3, 2)

conv\_layer3 = self.conv\_layer(pool\_layer2, 3, 384, 1, "Layer3")

conv\_layer4 = self.conv\_layer(conv\_layer3, 3, 384, 1, "Layer4")

conv\_layer5 = self.conv\_layer(conv\_layer4, 3, 256, 1, "Layer5")

pool\_layer = self.pool\_layer(conv\_layer5, 3, 2)

\_, x, y, z = pool\_layer.get\_shape()

full\_connect\_size = x \* y \* z

flatten = tf.reshape(pool\_layer, [-1, full\_connect\_size])

fc\_1 = self.full\_connect\_layer(flatten, 4096, "fc\_1")

drop1 = self.dropout(fc\_1, self.keep\_prob)

fc\_2 = self.full\_connect\_layer(drop1, 4096, "fc\_2")

drop2 = self.dropout(fc\_2, self.keep\_prob)

fc\_3 = self.full\_connect\_layer(drop2, self.num\_classes, "fc\_3")

return tf.nn.softmax(fc\_3)

def conv\_layer(self, X, ksize, out\_filters, stride, name):

in\_filters = int(X.get\_shape()[-1])

with tf.variable\_scope(name) as scope:

weight = tf.get\_variable("weight", [ksize, ksize, in\_filters, out\_filters])

bias = tf.get\_variable("bias", [out\_filters])

conv = tf.nn.conv2d(X, weight, strides=[1, stride, stride, 1], padding="SAME")

activation = tf.nn.relu(tf.nn.bias\_add(conv, bias))

return activation

#池化

def pool\_layer(self, X, ksize, stride):

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

#全连接

def full\_connect\_layer(self, X, out\_filters, name):

in\_filters = X.get\_shape()[-1]

with tf.variable\_scope(name) as scope:

w\_fc = tf.get\_variable("weight", shape=[in\_filters, out\_filters])

b\_fc = tf.get\_variable("bias", shape=[out\_filters], trainable=True)

fc = tf.nn.xw\_plus\_b(X, w\_fc, b\_fc)

return tf.nn.relu(fc)

#Dropout

def dropout(self, X, keep\_prob):

return tf.nn.dropout(X, keep\_prob)

#训练

def train\_val(X, y, y\_):

loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y, logits=y\_))

optimizer = tf.train.AdamOptimizer(learning\_rate=LR)

train\_op = optimizer.minimize(loss)

tf.summary.scalar("loss", loss)

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

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

merged = tf.summary.merge\_all()

writer = tf.summary.FileWriter("logs")

saver = tf.train.Saver()

with tf.Session() as sess:

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

writer.add\_graph(sess.graph)

i = 0

for epoch in range(EPOCHS):

for step in range(MAX\_STEPS):

batch\_xs, batch\_ys = mnist.train.next\_batch(BATCH\_SIZE)

summary, loss\_val, \_ = sess.run([merged, loss, train\_op],

feed\_dict={X: batch\_xs, y: batch\_ys})

print("epoch : {}----loss : {}".format(epoch, loss\_val))

writer.add\_summary(summary, i)

i += 1

saver.save(sess, os.path.join("temp", "mode.ckpt"))

test\_acc = 0

test\_count = 0

for \_ in range(10):

batch\_xs, batch\_ys = mnist.test.next\_batch(BATCH\_SIZE)

acc = sess.run(accuracy, feed\_dict={X: batch\_xs, y: batch\_ys})

test\_acc += acc

test\_count += 1

print("accuracy : {}".format(test\_acc / test\_count))

def main(\_):

X = tf.placeholder(tf.float32, [None, 784])

y = tf.placeholder(tf.float32, [BATCH\_SIZE, 10])

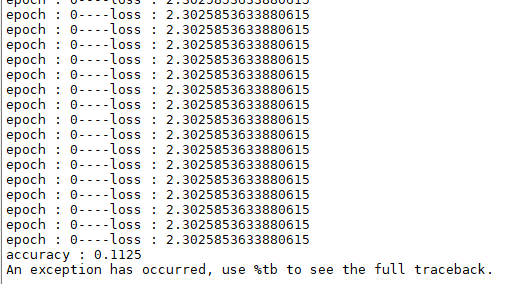
alex\_net = AlexNet(10, 0.5)

y\_ = alex\_net.create(X)

train\_val(X, y, y\_)

if \_\_name\_\_ == '\_\_main\_\_':

tf.app.run()



很有意思的是，发现效果非常差，原因在于：

同一个模型在不同的数据集上表现会存在很大差异，例如LeNet是在MNIST的基础上进行搭建和验证的，因此卷积核、步长等这些超参数都已经进行了精心的调节，因此只需要按照模型搭建完成即可得到99%以上的准确率。而AlexNet是在ImageNet的图像上进行调优的，ImageNet的图像相对于MNIST28\*28的图像要大很多，因此卷积核、步长都要大很多，但是这样对于图像较小的MNIST来说就相对较大，很难提取细节特征，因此如果用默认的结构效果甚至比不上20年轻的LeNet。这也是为什么深度学习模型可复制性差的原因，尽管是两个非常类似的任务，同一个模型在两个任务上表现得效果也会存在很大的差异，这需要工程时对其进行反复的调节、优化。

Tensorboard：

