1. 文件读写-TFRecord

class TFRecords\_Reader(object):

def \_\_init\_\_(self,num\_examples):

self.\_\_num\_examples = num\_examples

def write\_records(self,img\_dir=None,points\_dir=None,records\_name=None):

points\_txt = open(points\_dir, 'r')

writer = tf.python\_io.TFRecordWriter(records\_name)

for i in range(self.\_\_num\_examples):

j=int(points\_txt.readline())

x=float(points\_txt.readline())

y=float(points\_txt.readline())

img\_path = img\_dir + str(j) + '.jpg'

img = Image.open(img\_path)

img\_raw = img.tobytes()

example = tf.train.Example(features=tf.train.Features(feature={

"point\_x" : tf.train.Feature(float\_list=tf.train.FloatList(value=[x])),

"point\_y": tf.train.Feature(float\_list=tf.train.FloatList(value=[y])),

"index": tf.train.Feature(float\_list=tf.train.FloatList(value=[j])),

"img\_row" : tf.train.Feature(bytes\_list=tf.train.BytesList(value=[img\_raw]))

}))

writer.write(example.SerializeToString())

writer.close()

def readbatch\_by\_queue(self,records\_name=None,batch\_size=None,num\_epoch=None):

filename\_queue = tf.train.string\_input\_producer([records\_name],num\_epoch)

reader = tf.TFRecordReader()

\_, serialized\_example = reader.read(filename\_queue)

features = tf.parse\_single\_example(serialized\_example,

features={ "point\_x" : tf.FixedLenFeature([],tf.float32),

"point\_y": tf.FixedLenFeature([], tf.float32),

"index": tf.FixedLenFeature([], tf.float32),

"img\_row" : tf.FixedLenFeature([],tf.string)

})

img = tf.decode\_raw(features["img\_row"],tf.uint8)

img = tf.reshape(img,[224,224,1])

img = tf.cast(img, tf.float32)

points\_x = features["point\_x"]

points\_y = features["point\_y"]

index = features["index"]

min\_after\_dequeue = np.mod(self.\_\_num\_examples, batch\_size)

img\_batch, x\_batch,y\_batch,i\_batch = tf.train.shuffle\_batch(

[img, points\_x, points\_y,index], batch\_size=batch\_size,

capacity=self.\_\_num\_examples, min\_after\_dequeue=min\_after\_dequeue)

x\_batch = tf.expand\_dims(x\_batch,1)

y\_batch = tf.expand\_dims(y\_batch,1)

return img\_batch, x\_batch, y\_batch,i\_batch

1. 模型存储-save, np文件读写操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*读取路径，保存模型名\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

premodel\_dir = './newmodel\_6 /'

newmodel='newmodel\_7/model.ckpt'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*检查模型存在，并恢复\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ckpt = tf.train.get\_checkpoint\_state(premodel\_dir)

if ckpt and ckpt.model\_checkpoint\_path:

print('loading\_model')

saver = tf.train.Saver()

saver.restore(sess, ckpt.model\_checkpoint\_path)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*保存模型\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

saver.save(sess, newmodel, global\_step=j+1)

1. cnn常用函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*placeholder\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In = Tf.placeholder( dtype =tf.float32, shape= [None, 300, 400, 3])

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*conv+relu+max\_pool\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

W = tf.Variable(tf.truncated\_normal([11,11,3,96], stddev=0.1))//截断高斯

b = tf.Variable(tf.constant(0.1, shape=[96]))

conv1 = tf.nn.conv2d(c7, W, strides= [1, 4, 4, 1], padding='SAME')

relu1 = tf.nn.relu(conv1 + b)

maxpool1 = tf.nn.max\_pool(relu1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding=’SAME’)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*优化器\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

train\_step = tf.tra

in.AdadeltaOptimizer(0.5).minimize(loss\_L2)

1. 可视化tensorboard

#添每层的输出，四通道，可视化卷积层等

tf.summary.image("c7relu ", c7\_relu[:,:,:,25:28])

#可视化参数

tf.summary.histogram(layer\_name+'/weights', Weights)

#可视化loss

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

#合并

merged = tf.summary.merge\_all()

writer = tf.summary.FileWriter('log/',sess.graph)

#写入

ternel, merge = sess.run([x32\_norm, merged],feed\_dict={im\_origin: img\_array})

writer.add\_summary(merge)

6、冻结网络

1） 设置参数trainable = False

2） tf.stop\_gradients()，阻止相关参数反向传递

3）在外部运行完，将冻结网络的输出结果feed进训练网络，参考DQN的target和eval网络的方式

4）选择某一个collection集合的参数优化

The easiest way to achieve this, as you mention in your question, is to create two optimizer operations using separate calls to opt.minimize(cost, ...). By default, the optimizer will use all of the variables in [tf.trainable\_variables()](https://www.tensorflow.org/versions/master/api_docs/python/state_ops.html#trainable_variables). If you want to filter the variables to a particular scope, you can use the optional scope argument to [tf.get\_collection()](https://www.tensorflow.org/versions/master/api_docs/python/framework.html#get_collection) as follows:

optimizer = tf.train.AdagradOptimzer(0.01)

first\_train\_vars = tf.get\_collection(tf.GraphKeys.TRAINABLE\_VARIABLES,

"scope/prefix/for/first/vars")

first\_train\_op = optimizer.minimize(cost, var\_list=first\_train\_vars)

second\_train\_vars = tf.get\_collection(tf.GraphKeys.TRAINABLE\_VARIABLES,

"scope/prefix/for/second/vars")

second\_train\_op = optimizer.minimize(cost, var\_list=second\_train\_vars)