目录

[1 文件读写-TFRecord 2](#_Toc499112855)

[2 模型存储-save/restore 4](#_Toc499112856)

[3 cnn常用函数 5](#_Toc499112857)

[4 可视化tensorboard 6](#_Toc499112858)

[5 冻结网络 7](#_Toc499112859)

[6 图像预处理 7](#_Toc499112860)

[7 其他： 8](#_Toc499112861)

[7.1 graph: 8](#_Toc499112862)

[7.2 队列与多线程：未整理 8](#_Toc499112863)

[7.3 多GPU编程：未整理 8](#_Toc499112864)

[7.4 name\_scope: 8](#_Toc499112865)

[7.5 collection 8](#_Toc499112866)

# 文件读写-TFRecord

import tensorflow as tf  
from PIL import Image  
  
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):  
 #将文件存成多个records，然后读取  
 #files = tf.train.match\_filenames\_once(‘’/data/tfrecords\*)  
 #filename\_queue = tf.train.string\_input\_producer(files,num\_epoch)  
 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)  
 #后边接图像预处理，可以借助shuffle\_batch中的多线程  
 # img = preprocess(img) #添加随机性  
 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)  
 # capacit为队列容量，队列不满时才从文件中执行入队操作；  
 # min\_after\_dequeue为队列最少存在的个数，否则随机效果不好，队列元素小于batch则等待入队操作后在出队， 可是设为1000或10000；  
 # num\_threads可以指定线程数, 默认为1个；  
 img\_batch, x\_batch,y\_batch,i\_batch = tf.train.shuffle\_batch(  
 [img, points\_x, points\_y,index], batch\_size=batch\_size,  
 capacity= min\_after\_dequeue +3\*batch\_size, 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

coord = tf.train.Coordinator()  
threads = tf.train.start\_queue\_runners(sess=sess, coord=coord)  
*try*:  
 *while not* coord.should\_stop():  
 j = j + 1  
 index\_batch, image\_batch, annot\_image\_batch = sess.run([index, image, annot\_image])  
 sess.run(train\_step, feed\_dict={im\_origin: image\_batch, annot\_im: annot\_image\_batch})  
 *if* (j + 1) % SAVER\_STEP == 0:  
 saver.save(sess, tfmodel\_name, global\_step=j + 1)  
*except* tf.errors.OutOfRangeError:  
 print('Done training')  
*finally*:  
 coord.request\_top()  
  
coord.join(threads)  
  
sess.close()

# 模型存储-save/restore

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*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)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*保存部分参数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
a = tf.Variable( tf.constant(0, shape=[1]), name='a', collections=[ 'save', tf.GraphKeys.GLOBAL\_VARIABLES])  
b = tf.Variable( tf.constant(1, shape=[2]), name='b',collections=[ 'save', tf.GraphKeys.GLOBAL\_VARIABLES])  
c = tf.Variable(tf.constant(2, shape=[3]), name='c')  
  
*with* tf.Session() *as* sess:  
 sess.run(tf.global\_variables\_initializer())  
 saver = tf.train.Saver(var\_list=tf.get\_collection('save'))  
 saver.save(sess, 'model/model.ckpt')  
#\*\*\*\*\*\*\*\*\*\*加载部分参数，重映射到不同名字的变量， 尺寸必须相同\*\*\*\*\*\*\*\*\*\*  
ab = tf.Variable( tf.constant(2, shape=[1]), name='ab')  
bb = tf.Variable( tf.constant(10, shape=[2]), name='bb')  
cb = tf.Variable(tf.constant(2, shape=[3]), name='cb')  
  
*with* tf.Session() *as* sess:  
 sess.run(tf.global\_variables\_initializer())  
 saver = tf.train.Saver({'a': ab })  
 ckpt = tf.train.get\_checkpoint\_state('model/')  
 saver.restore(sess, ckpt.model\_checkpoint\_path)  
 print(sess.run(ab))  
 print(sess.run(bb))

# cnn常用函数

Learning\_rate\_base = 0.5  
batch\_size = 32  
Num\_examples = 5000   
Learning\_rate\_decay = 0.99  
  
#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*placeholder\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
input = tf.placeholder( dtype =tf.float32, shape= [*None*, 300, 400, 3])  
  
#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*conv+relu+max\_pool\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\_lambda = 0.001  
W = tf.Variable(tf.truncated\_normal([11,11,3,96], stddev=0.1))#截断高斯  
tf.add\_to\_collection('L2\_loss', tf.contrib.layers.l2\_regularizer(\_lambda)(W))  
b = tf.Variable(tf.constant(0.1, shape=[96]))  
tf.add\_to\_collection('L2\_loss', tf.contrib.layers.l2\_regularizer(\_lambda)(b))  
conv1 = tf.nn.conv2d(input, 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')  
  
loss = #……..# 计算loss  
tf.add\_to\_collection('L2\_loss', loss) #收集名称为‘L2\_Loss’的变量，无论变量在Global\_variables中还是在其他的key中  
  
#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*优化器\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

global\_step = tf.get\_variable('global\_step', [], initializer=tf.constant\_initializer(0), trainable=*False*)  
Learning\_rate= tf.train.exponential\_decay(  
 learning\_rate\_base,  
 global\_step,  
 num\_examples / batch\_size,#训练完一次样本需要多少轮次，  
 learning\_rate\_decay,#0.99  
)  
loss\_L2 =tf.add\_n(tf.get\_collection('L2\_loss'))  
train\_step = tf.train.AdadeltaOptimizer(Learning\_rate).minimize(loss\_L2)

# 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)

**可以显示节点在某个时间戳上的时间和空间消耗。**

*if* train\_step % 1000 ==0 :  
 test\_input = mnist.test.images[:1000]  
 test\_target = mnist.test.labels[:1000]  
 run\_options = tf.RunOptions(trace\_level=tf.RunOptions.FULL\_TRACE)  
 run\_metadata = tf.RunMetadata()  
 \_,\_,\_,net\_test\_sum = sess.run([net.accuracy\_without\_bn, net.accuracy\_bn\_before\_relu, net.accuracy\_bn\_after\_relu, net.merge\_test],  
 feed\_dict={net.input: test\_input, net.label: test\_target, net.keep\_prob: 1, net.is\_train:*False*},  
 options=run\_options, run\_metadata=run\_metadata)  
 tf\_sum\_writer.add\_run\_metadata(run\_metadata, 'step%03d' % train\_step)  
 tf\_sum\_writer.add\_summary(net\_test\_sum, test\_step)  
 test\_step += 1

# 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

#定义参数，以及其所在的collection  
collection\_name = ['train\_0', tf.GraphKeys.GLOBAL\_VARIABLES]  
w = tf.Variable(tf.constant(0, shape=[10]), collections = collection\_name)  
# tf.add\_to\_collection(‘train\_0’, w) #将w添加进collection, 此时该collection包含两个w  
  
#选择需要优化的参数进行优化  
optimizer = tf.train.AdagradOptimzer(0.01)  
first\_train\_vars = tf.get\_collection('train\_0')  
first\_train\_op = optimizer.minimize(cost, var\_list=first\_train\_vars)

# 图像预处理

**用于图像增强，增加图像的随机性**

#旋转，翻转，大小  
#色彩调整  
tf.image.adjust\_brightness(img\_data, 0.1) #亮度+0.1  
tf.image.random\_brightness(img\_data, 0.1) # -0.1 ~ 0.1之间随意调整对比度  
tf.image.adjust\_contrast #对比度  
tf.image.adjust\_hue #色调  
tf.image.adjust\_saturation #饱和度  
#裁剪  
Begin,size,bbox\_for\_draw = tf.image.sample\_distorted\_bounding\_bos(tf.shape(img\_data), bounding\_boxes=boxes) #boxes为相对大小，0-1之间的数，左上角，右下角坐标  
Distorted\_imag = tf.slice(img\_data, begin, size) # 裁剪后数据，img\_data为四维（有batch）  
Img\_with\_box\_for\_draw = tf.image.draw\_bounding\_box(img\_data, bb0x\_for\_draw) #用以显示的图像，带框

# 其他：

## graph:

* 默认graph

Tf.get\_default\_graph(), 可得到默认graph的信息；

未指定graph, 则变量及sess运行默认graph

* 构建新的graph

g = tf.Graph()  
# 设置 graph, 运行的设备  
*with* g.as\_default(), g.device('/gpu:0'):  
 #定义变量  
 b = tf.Variable(tf.constant(0., shape=[6]))  
 #打印 变量b所属的graph信息  
 print(b.graph)  
 print(g)  
   
# 指定sess运行的graph  
*with* tf.Session(graph=g) *as* sess:  
 sess.run(tf.global\_variables\_initializer())

## 队列与多线程：未整理

## 多GPU编程：未整理

## name\_scope:

## collection

a = tf.Variable(tf.constant(0.1,shape=[1]),

collections=['loss',tf.GraphKeys.MOVING\_AVERAGE\_VARIABLES]) #不被sess.run(tf.global\_variables\_initializer()初始化，要使用sess.run(a.initializer进行初始化)或sess.run(tf.variables\_initializer(tf.moving\_average\_variables()))

tf.add\_to\_collection(‘name’,value) #添加变量到‘name’,无论value属于哪个Key

tf.get\_collection(‘name’, ‘scope’) #获取‘name’的collection中的变量列表，可以具体到name\_scope为‘scope’的变量，名为‘name’的collection可以包含global\_variable, moving\_average\_variable等不同Key的变量