一 引入

Tfrecord是bin文件,有几个有点:其一,清晰易用; 其二,减少访问硬盘的耗时,因为图片和注释放在一起了,而不是单独摆放的.

实验一 获得图片的raw数据

|  |
| --- |
| **%**matplotlib inline  import numpy **as** np  import skimage.io **as** io  cat\_img **=** io**.**imread('cat.jpg')  io**.**imshow(cat\_img)  png  *# Let's convert the picture into string representation*  *# using the ndarray.tostring() function*  cat\_string **=** cat\_img**.**tostring()  *# Now let's convert the string back to the image*  *# Important: the dtype should be specified*  *# otherwise the reconstruction will be errorness*  *# Reconstruction is 1d, so we need sizes of image*  *# to fully reconstruct it.*  reconstructed\_cat\_1d **=** np**.**fromstring(cat\_string, dtype**=**np**.**uint8)  *# Here we reshape the 1d representation*  *# This is the why we need to store the sizes of image*  *# along with its serialized representation.*  reconstructed\_cat\_img **=** reconstructed\_cat\_1d**.**reshape(cat\_img**.**shape)  *# Let's check if we got everything right and compare*  *# reconstructed array to the original one.*  np**.**allclose(cat\_img, reconstructed\_cat\_img) |

先转换成string类型.

如果想要把string类型再次转回图片的话,需要知道dtype,然后再按照原图的尺寸转回去.

最后利用np.allclose来检查两图是否一致.

注意这里需要原图的size.这个很有必要.

实验二 创建tfrecord并在不建立graph情况下读取它.

|  |
| --- |
| *# Get some image/annotation pairs for example*  filename\_pairs **=** [  ('/home/dpakhom1/tf\_projects/segmentation/VOCdevkit/VOCdevkit/VOC2012/JPEGImages/2007\_000032.jpg',  '/home/dpakhom1/tf\_projects/segmentation/VOCdevkit/VOCdevkit/VOC2012/SegmentationClass/2007\_000032.png'),  ('/home/dpakhom1/tf\_projects/segmentation/VOCdevkit/VOCdevkit/VOC2012/JPEGImages/2007\_000039.jpg',  '/home/dpakhom1/tf\_projects/segmentation/VOCdevkit/VOCdevkit/VOC2012/SegmentationClass/2007\_000039.png'),  ('/home/dpakhom1/tf\_projects/segmentation/VOCdevkit/VOCdevkit/VOC2012/JPEGImages/2007\_000063.jpg',  '/home/dpakhom1/tf\_projects/segmentation/VOCdevkit/VOCdevkit/VOC2012/SegmentationClass/2007\_000063.png')  ] |

|  |
| --- |
| **%**matplotlib inline  *# Important: We are using PIL to read .png files later.*  *# This was done on purpose to read indexed png files*  *# in a special way -- only indexes and not map the indexes*  *# to actual rgb values. This is specific to PASCAL VOC*  *# dataset data. If you don't want thit type of behaviour*  *# consider using skimage.io.imread()*  from PIL import Image  import numpy **as** np  import skimage.io **as** io  import tensorflow **as** tf  **def** **\_bytes\_feature**(value):  **return** tf**.**train**.**Feature(bytes\_list**=**tf**.**train**.**BytesList(value**=**[value]))  **def** **\_int64\_feature**(value):  **return** tf**.**train**.**Feature(int64\_list**=**tf**.**train**.**Int64List(value**=**[value]))  tfrecords\_filename **=** 'pascal\_voc\_segmentation.tfrecords'  writer **=** tf**.**python\_io**.**TFRecordWriter(tfrecords\_filename)  *# Let's collect the real images to later on compare*  *# to the reconstructed ones*  original\_images **=** []  **for** img\_path, annotation\_path **in** filename\_pairs:    img **=** np**.**array(Image**.**open(img\_path))  annotation **=** np**.**array(Image**.**open(annotation\_path))    *# The reason to store image sizes was demonstrated*  *# in the previous example -- we have to know sizes*  *# of images to later read raw serialized string,*  *# convert to 1d array and convert to respective*  *# shape that image used to have.*  height **=** img**.**shape[0]  width **=** img**.**shape[1]    *# Put in the original images into array*  *# Just for future check for correctness*  original\_images**.**append((img, annotation))    img\_raw **=** img**.**tostring()  annotation\_raw **=** annotation**.**tostring()    example **=** tf**.**train**.**Example(features**=**tf**.**train**.**Features(feature**=**{  'height': \_int64\_feature(height),  'width': \_int64\_feature(width),  'image\_raw': \_bytes\_feature(img\_raw),  'mask\_raw': \_bytes\_feature(annotation\_raw)}))    writer**.**write(example**.**SerializeToString())  writer**.**close() |

保存到tfrecord.利用tf.train.Example以及tf.train.Features方法.

|  |
| --- |
| reconstructed\_images **=** []  record\_iterator **=** tf**.**python\_io**.**tf\_record\_iterator(path**=**tfrecords\_filename)  **for** string\_record **in** record\_iterator:    example **=** tf**.**train**.**Example()  example**.**ParseFromString(string\_record)    height **=** int(example**.**features**.**feature['height']  **.**int64\_list  **.**value[0])    width **=** int(example**.**features**.**feature['width']  **.**int64\_list  **.**value[0])    img\_string **=** (example**.**features**.**feature['image\_raw']  **.**bytes\_list  **.**value[0])    annotation\_string **=** (example**.**features**.**feature['mask\_raw']  **.**bytes\_list  **.**value[0])    img\_1d **=** np**.**fromstring(img\_string, dtype**=**np**.**uint8)  reconstructed\_img **=** img\_1d**.**reshape((height, width, **-**1))    annotation\_1d **=** np**.**fromstring(annotation\_string, dtype**=**np**.**uint8)    *# Annotations don't have depth (3rd dimension)*  reconstructed\_annotation **=** annotation\_1d**.**reshape((height, width))    reconstructed\_images**.**append((reconstructed\_img, reconstructed\_annotation)) |

从tfrecord文件中读出信息并重建图片.

|  |
| --- |
| *# Let's check if the reconstructed images match*  *# the original images*  **for** original\_pair, reconstructed\_pair **in** zip(original\_images, reconstructed\_images):    img\_pair\_to\_compare, annotation\_pair\_to\_compare **=** zip(original\_pair,  reconstructed\_pair)  **print**(np**.**allclose(**\***img\_pair\_to\_compare))  **print**(np**.**allclose(**\***annotation\_pair\_to\_compare)) |

对比重构图和原图的一致性.分别比较图片和注释图片.

具体的结构如下:



实验三 在Graph下对tfrecord的图片进行批量读取

|  |
| --- |
| **%**matplotlib inline  import tensorflow **as** tf  import skimage.io **as** io  IMAGE\_HEIGHT **=** 384  IMAGE\_WIDTH **=** 384  tfrecords\_filename **=** 'pascal\_voc\_segmentation.tfrecords'  **def** **read\_and\_decode**(filename\_queue):    reader **=** tf**.**TFRecordReader()  \_, serialized\_example **=** reader**.**read(filename\_queue)  features **=** tf**.**parse\_single\_example(  serialized\_example,  *# Defaults are not specified since both keys are required.*  features**=**{  'height': tf**.**FixedLenFeature([], tf**.**int64),  'width': tf**.**FixedLenFeature([], tf**.**int64),  'image\_raw': tf**.**FixedLenFeature([], tf**.**string),  'mask\_raw': tf**.**FixedLenFeature([], tf**.**string)  })  *# Convert from a scalar string tensor (whose single string has*  *# length mnist.IMAGE\_PIXELS) to a uint8 tensor with shape*  *# [mnist.IMAGE\_PIXELS].*  image **=** tf**.**decode\_raw(features['image\_raw'], tf**.**uint8)  annotation **=** tf**.**decode\_raw(features['mask\_raw'], tf**.**uint8)    height **=** tf**.**cast(features['height'], tf**.**int32)  width **=** tf**.**cast(features['width'], tf**.**int32)    image\_shape **=** tf**.**pack([height, width, 3])  annotation\_shape **=** tf**.**pack([height, width, 1])    image **=** tf**.**reshape(image, image\_shape)  annotation **=** tf**.**reshape(annotation, annotation\_shape)    image\_size\_const **=** tf**.**constant((IMAGE\_HEIGHT, IMAGE\_WIDTH, 3), dtype**=**tf**.**int32)  annotation\_size\_const **=** tf**.**constant((IMAGE\_HEIGHT, IMAGE\_WIDTH, 1), dtype**=**tf**.**int32)    *# Random transformations can be put here: right before you crop images*  *# to predefined size. To get more information look at the stackoverflow*  *# question linked above.*    resized\_image **=** tf**.**image**.**resize\_image\_with\_crop\_or\_pad(image**=**image,  target\_height**=**IMAGE\_HEIGHT,  target\_width**=**IMAGE\_WIDTH)    resized\_annotation **=** tf**.**image**.**resize\_image\_with\_crop\_or\_pad(image**=**annotation,  target\_height**=**IMAGE\_HEIGHT,  target\_width**=**IMAGE\_WIDTH)      images, annotations **=** tf**.**train**.**shuffle\_batch( [resized\_image, resized\_annotation],  batch\_size**=**2,  capacity**=**30,  num\_threads**=**2,  min\_after\_dequeue**=**10)    **return** images, annotations |

提供一个批量操作tfrecord的函数.

|  |
| --- |
| filename\_queue **=** tf**.**train**.**string\_input\_producer(  [tfrecords\_filename], num\_epochs**=**10)  *# Even when reading in multiple threads, share the filename*  *# queue.*  image, annotation **=** read\_and\_decode(filename\_queue)  *# The op for initializing the variables.*  init\_op **=** tf**.**group(tf**.**global\_variables\_initializer(),  tf**.**local\_variables\_initializer())  **with** tf**.**Session() **as** sess:    sess**.**run(init\_op)    coord **=** tf**.**train**.**Coordinator()  threads **=** tf**.**train**.**start\_queue\_runners(coord**=**coord)    *# Let's read off 3 batches just for example*  **for** i **in** xrange(3):    img, anno **=** sess**.**run([image, annotation])  **print**(img[0, :, :, :]**.**shape)    **print**('current batch')    *# We selected the batch size of two*  *# So we should get two image pairs in each batch*  *# Let's make sure it is random*  io**.**imshow(img[0, :, :, :])  io**.**show()  io**.**imshow(anno[0, :, :, 0])  io**.**show()    io**.**imshow(img[1, :, :, :])  io**.**show()  io**.**imshow(anno[1, :, :, 0])  io**.**show()      coord**.**request\_stop()  coord**.**join(threads) |

对之前构建的tfrecord文件进行batch读取,每次读取2组,每组包含img和annotation.