姓名：向建宇

学号：1801210687

**采用一种Inception之外的其它模型如Alexnet/GoogleNet/ResNet，分析flowers\_photos数据集，得出准确率。**

这次作业我用的是vgg16来完成的，因为它的层数比较少，只有16层，采用了<https://github.com/tensorflow/models/tree/master/research/slim> 上训练好了的ckpt，只是将最后的全连接层参数进行了重新训练。

1.**处理图片：**

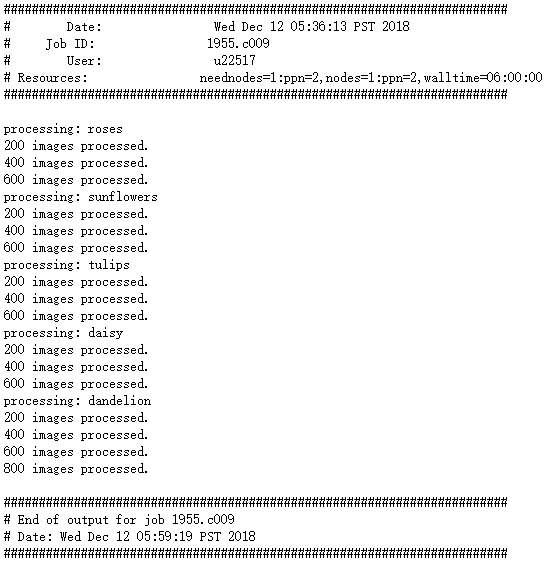
Vgg16的输入图片是224\*224分辨率的，所以第一步要将flowers处理成对应的分辨率并用npy保存起来。

首先要读取数据，并把数据分成训练集、测试集和验证集。

这里我首先在本地进行处理，我的电脑是8g内存的，跑了接近2个小时，依然报了内存不足的错误

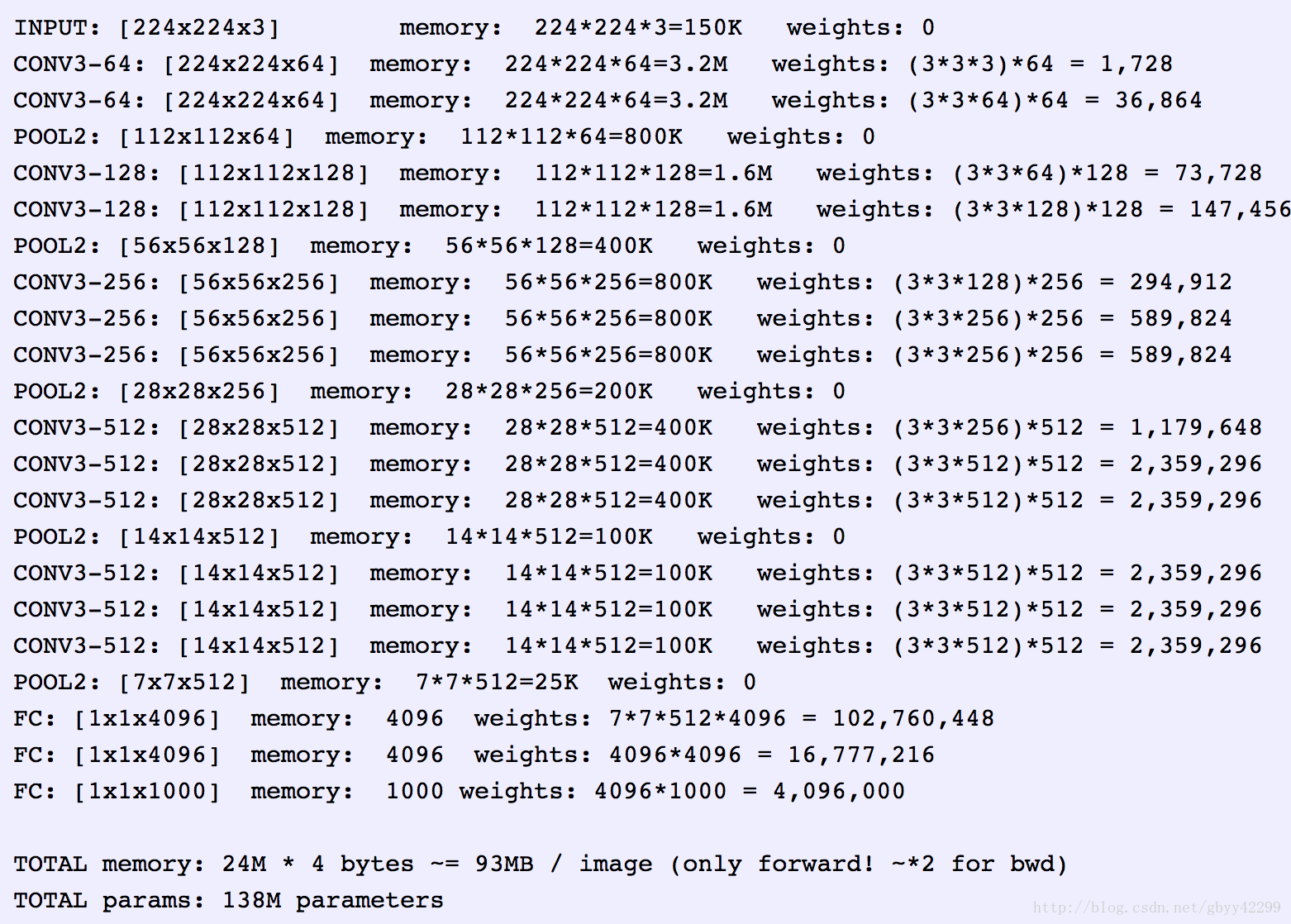


没办法，想到了英特尔AI cloud，结果20min左右就处理完毕，性能还是很强大的。最后看返回文件的输出：



**2.vgg\_16：**

它的网络结构如下所示：



1、输入224x224x3的图片，经过64个卷积核的两次卷积后，采用一次pooling。经过第一次卷积后，c1有（3x3x3）个可训练参数

2、之后又经过两次128的卷积核卷积之后，采用一次pooling

3、再经过三次256的卷积核的卷积之后，采用pooling

4、重复两次三个512的卷积核卷积之后再pooling。

5、三次Fc

**3.使用vgg\_16的ckpt进行迁移学习：**

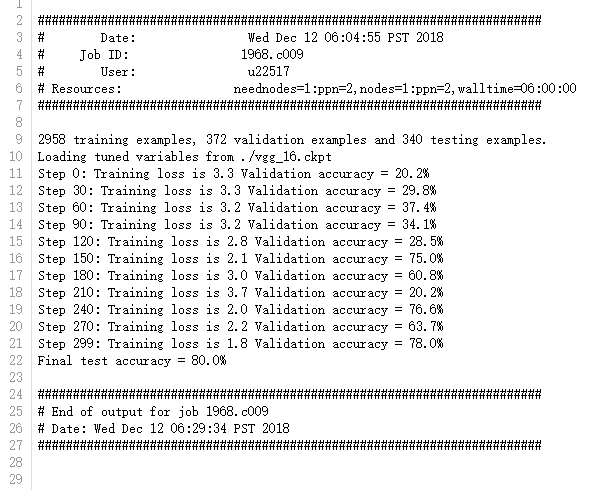
从上面可以看出最后三个是全连接层fc，需要我们自己来训练，所以查看vgg的code源码发现

这三层的名称分别为fc6、fc7、fc8，后面要

|  |
| --- |
| # 不需要从谷歌训练好的模型中加载的参数。  CHECKPOINT\_EXCLUDE\_SCOPES = 'vgg\_16/fc6,vgg\_16/fc7,vgg\_16/fc8'  # 需要训练的网络层参数明层，在fine-tuning的过程中就是最后的全联接层。  TRAINABLE\_SCOPES='vgg\_16/fc6,vgg\_16/fc7,vgg\_16/fc8' |

并对照github的源码修改其它相应的参数

4.结果截图：



使用了老师给的flowers数据集，共五种类型的花

2958个训练集，372验证集，340测试集，最后正确率在80%，

对比官网的71.5，主要是自己的类太少只有5类，而官网上为1000个类



附录一：（图片预处理）

|  |
| --- |
| #1.定义需要使用到的常量  import glob  import os.path  import numpy as np  import tensorflow as tf  from tensorflow.python.platform import gfile  # 原始输入数据的目录，这个目录下有5个子目录，每个子目录底下保存这属于该  # 类别的所有图片。  INPUT\_DATA = './flower\_photos'  # 输出文件地址。我们将整理后的图片数据通过numpy的格式保存。  OUTPUT\_FILE = './flower\_processed\_data.npy'  # 测试数据和验证数据比例。  VALIDATION\_PERCENTAGE = 10  TEST\_PERCENTAGE = 10  #2.定义数据处理过程  # 读取数据并将数据分割成训练数据、验证数据和测试数据。  def create\_image\_lists(sess, testing\_percentage, validation\_percentage):  sub\_dirs = [x[0] for x in os.walk(INPUT\_DATA)]  is\_root\_dir = True  # 初始化各个数据集。  training\_images = []  training\_labels = []  testing\_images = []  testing\_labels = []  validation\_images = []  validation\_labels = []  current\_label = 0  # 读取所有的子目录。  for sub\_dir in sub\_dirs:  if is\_root\_dir:  is\_root\_dir = False  continue  # 获取一个子目录中所有的图片文件。  extensions = ['jpg', 'jpeg', 'JPG', 'JPEG']  file\_list = []  dir\_name = os.path.basename(sub\_dir)  for extension in extensions:  file\_glob = os.path.join(INPUT\_DATA, dir\_name, '\*.' + extension)  file\_list.extend(glob.glob(file\_glob))  if not file\_list: continue  print("processing:", dir\_name)  i = 0  # 处理图片数据。  for file\_name in file\_list:  i += 1  # 读取并解析图片，将图片转化为299\*299以方便inception-v3模型来处理。  image\_raw\_data = gfile.FastGFile(file\_name, 'rb').read()  image = tf.image.decode\_jpeg(image\_raw\_data)  if image.dtype != tf.float32:  image = tf.image.convert\_image\_dtype(image, dtype=tf.float32)  image = tf.image.resize\_images(image, [224, 224])  image\_value = sess.run(image)  # 随机划分数据聚。  chance = np.random.randint(100)  if chance < validation\_percentage:  validation\_images.append(image\_value)  validation\_labels.append(current\_label)  elif chance < (testing\_percentage + validation\_percentage):  testing\_images.append(image\_value)  testing\_labels.append(current\_label)  else:  training\_images.append(image\_value)  training\_labels.append(current\_label)  if i % 200 == 0:  print(i, "images processed.")  current\_label += 1  # 将训练数据随机打乱以获得更好的训练效果。  state = np.random.get\_state()  np.random.shuffle(training\_images)  np.random.set\_state(state)  np.random.shuffle(training\_labels)  return np.asarray([training\_images, training\_labels,  validation\_images, validation\_labels,  testing\_images, testing\_labels])  #3.运行数据处理过程  with tf.Session() as sess:  processed\_data = create\_image\_lists(sess, TEST\_PERCENTAGE, VALIDATION\_PERCENTAGE)  # 通过numpy格式保存处理后的数据。  np.save(OUTPUT\_FILE, processed\_data) |

附录二：（迁移学习）

|  |
| --- |
| #1. 定义训练过程中将要使用到的常量。  #因为GitHub无法保存大于100M的文件，所以在运行时需要先自行从Google下载vgg\_16.ckpt文件。  import glob  import os.path  import numpy as np  import tensorflow as tf  from tensorflow.python.platform import gfile  import tensorflow.contrib.slim as slim  # 加载通过TensorFlow-Slim定义好的inception\_v3模型。  import tensorflow.contrib.slim.python.slim.nets.vgg as vgg\_16  # 处理好之后的数据文件。  INPUT\_DATA = './flower\_processed\_data.npy'  # 保存训练好的模型的路径。  TRAIN\_FILE = './train\_dir/model'  # 谷歌提供的训练好的模型文件地址。因为GitHub无法保存大于100M的文件，所以  # 在运行时需要先自行从Google下载vgg\_16.ckpt文件。  CKPT\_FILE = './vgg\_16.ckpt'  # 定义训练中使用的参数。  LEARNING\_RATE = 0.0001  STEPS = 300  BATCH = 32  N\_CLASSES = 5  # 不需要从谷歌训练好的模型中加载的参数。  CHECKPOINT\_EXCLUDE\_SCOPES = 'vgg\_16/fc6,vgg\_16/fc7,vgg\_16/fc8'  # 需要训练的网络层参数明层，在fine-tuning的过程中就是最后的全联接层。  TRAINABLE\_SCOPES='vgg\_16/fc6,vgg\_16/fc7,vgg\_16/fc8'  #2. 获取所有需要从谷歌训练好的模型中加载的参数。  def get\_tuned\_variables():  exclusions = [scope.strip() for scope in CHECKPOINT\_EXCLUDE\_SCOPES.split(',')]  variables\_to\_restore = []  # 枚举vgg模型中所有的参数，然后判断是否需要从加载列表中移除。  for var in slim.get\_model\_variables():  excluded = False  for exclusion in exclusions:  if var.op.name.startswith(exclusion):  excluded = True  break  if not excluded:  variables\_to\_restore.append(var)  return variables\_to\_restore  #3. 获取所有需要训练的变量列表。  def get\_trainable\_variables():  scopes = [scope.strip() for scope in TRAINABLE\_SCOPES.split(',')]  variables\_to\_train = []  # 枚举所有需要训练的参数前缀，并通过这些前缀找到所有需要训练的参数。  for scope in scopes:  variables = tf.get\_collection(tf.GraphKeys.TRAINABLE\_VARIABLES, scope)  variables\_to\_train.extend(variables)  return variables\_to\_train  #4. 定义训练过程。  def main():  # 加载预处理好的数据。  processed\_data = np.load(INPUT\_DATA)  training\_images = processed\_data[0]  n\_training\_example = len(training\_images)  training\_labels = processed\_data[1]  validation\_images = processed\_data[2]  validation\_labels = processed\_data[3]  testing\_images = processed\_data[4]  testing\_labels = processed\_data[5]  print("%d training examples, %d validation examples and %d testing examples." % (  n\_training\_example, len(validation\_labels), len(testing\_labels)))  # 定义vgg的输入，images为输入图片，labels为每一张图片对应的标签。  images = tf.placeholder(tf.float32, [None, 224, 224, 3], name='input\_images')  labels = tf.placeholder(tf.int64, [None], name='labels')  # 定义vgg模型。因为谷歌给出的只有模型参数取值，所以这里  # 需要在这个代码中定义vgg的模型结构。虽然理论上需要区分训练和  # 测试中使用到的模型，也就是说在测试时应该使用is\_training=False，但是  # 因为预先训练好的vgg模型中使用的batch normalization参数与  # 新的数据会有出入，所以这里直接使用同一个模型来做测试。  with slim.arg\_scope(vgg\_16.vgg\_arg\_scope()):  logits, \_ = vgg\_16.vgg\_16(  images, num\_classes=N\_CLASSES, is\_training=True)  trainable\_variables = get\_trainable\_variables()  # 定义损失函数和训练过程。  tf.losses.softmax\_cross\_entropy(  tf.one\_hot(labels, N\_CLASSES), logits, weights=1.0)  total\_loss = tf.losses.get\_total\_loss()  train\_step = tf.train.RMSPropOptimizer(LEARNING\_RATE).minimize(total\_loss)  # 计算正确率。  with tf.name\_scope('evaluation'):  correct\_prediction = tf.equal(tf.argmax(logits, 1), labels)  evaluation\_step = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  # 定义加载Google训练好的vgg模型的Saver。  load\_fn = slim.assign\_from\_checkpoint\_fn(  CKPT\_FILE,  get\_tuned\_variables(),  ignore\_missing\_vars=True)  # 定义保存新模型的Saver。  saver = tf.train.Saver()  with tf.Session() as sess:  # 初始化没有加载进来的变量。  init = tf.global\_variables\_initializer()  sess.run(init)  # 加载谷歌已经训练好的模型。  print('Loading tuned variables from %s' % CKPT\_FILE)  load\_fn(sess)  start = 0  end = BATCH  for i in range(STEPS):  \_, loss = sess.run([train\_step, total\_loss], feed\_dict={  images: training\_images[start:end],  labels: training\_labels[start:end]})  if i % 30 == 0 or i + 1 == STEPS:  saver.save(sess, TRAIN\_FILE, global\_step=i)  validation\_accuracy = sess.run(evaluation\_step, feed\_dict={  images: validation\_images, labels: validation\_labels})  print('Step %d: Training loss is %.1f Validation accuracy = %.1f%%' % (  i, loss, validation\_accuracy \* 100.0))  start = end  if start == n\_training\_example:  start = 0  end = start + BATCH  if end > n\_training\_example:  end = n\_training\_example  # 在最后的测试数据上测试正确率。  test\_accuracy = sess.run(evaluation\_step, feed\_dict={  images: testing\_images, labels: testing\_labels})  print('Final test accuracy = %.1f%%' % (test\_accuracy \* 100))  #5. 运行训练过程。  if \_\_name\_\_ == '\_\_main\_\_':  main() |