<http://blog.csdn.net/u014432647/article/details/75276718>

这篇薄荷主要是讲了如何用tensorflow去训练好一个模型，然后生成相应的pb文件。最后会将如何重新加载这个pb文件。   
首先先放出PO主的github：

[https://github.com/ppplinday/tensorflow-vgg16-train-and-test](https://github.com/ppplinday/tensorflow-vgg16-train-and-test" \t "http://blog.csdn.net/u014432647/article/details/_blank)

其中的pitcute文件是狗和猫的图片分别15张一共30（别吐槽，只是为了练手学习的233333）， train那个就是训练的文件，test这个就是测试的文件。   
接着PO主会慢慢讲解相应的步骤。   
！！！ps：由于PO主也是新手，所以难免会出现一点（很多）小错误，希望大婶看了能够提出来让PO主好好学习233333。

1. train   
   首先说一下train。一开始当然是读图片啦。

|  |
| --- |
| def read\_img(path):  cate = [path + x for x in os.listdir(path) if os.path.isdir(path + x)]  imgs = []  labels = []  for idx, folder in enumerate(cate):  for im in glob.glob(folder + '/\*.jpg'):  print('reading the image: %s' % (im))  img = io.imread(im)  img = transform.resize(img, (w, h, c))  imgs.append(img)  labels.append(idx)  return np.asarray(imgs, np.float32), np.asarray(labels, np.int32)  data, label = read\_img(path) |

用io.imread来读取每一张图片，然后resize成vgg的输入的大小（224，224，3），最后分别放入了data和label中。

|  |
| --- |
| num\_example = data.shape[0]  arr = np.arange(num\_example)  np.random.shuffle(arr)  data = data[arr]  label = label[arr] |

这里是把图片的顺序打乱，先生成一个等差数列，然后打乱，最后赋值回原来的data和label

|  |
| --- |
| ratio = 0.8  s = np.int(num\_example \* ratio)  x\_train = data[:s]  y\_train = label[:s]  x\_val = data[s:]  y\_val = label[s:] |

全部的数据中百分之80的用来train，剩下20的用来test（虽然一共才30张图片。。。。。）

|  |
| --- |
| def build\_network(height, width, channel):  x = tf.placeholder(tf.float32, shape=[None, height, width, channel], name='input')  y = tf.placeholder(tf.int64, shape=[None, 2], name='labels\_placeholder') |

开始build相应的vgg model，这一步不难，但是每一层最好都给上相应的name。上面的x和y是相应的输入和相应的标签。

|  |
| --- |
| finaloutput = tf.nn.softmax(output\_fc8, name="softmax")  cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=finaloutput, labels=y))  optimize = tf.train.AdamOptimizer(learning\_rate=1e-4).minimize(cost)  prediction\_labels = tf.argmax(finaloutput, axis=1, name="output")  read\_labels = y  correct\_prediction = tf.equal(prediction\_labels, read\_labels)  accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  correct\_times\_in\_batch = tf.reduce\_sum(tf.cast(correct\_prediction, tf.int32))  return dict(  x=x,  y=y,  optimize=optimize,  correct\_prediction=correct\_prediction,  correct\_times\_in\_batch=correct\_times\_in\_batch,  cost=cost,  ) |

在build的最后，是需要进行误差计算。finaloutput是最后的输出，cost是计算误差，optimize是定义训练时候安什么方式，也注意一下最后的return。

接着是训练过程。

|  |
| --- |
| def train\_network(graph, batch\_size, num\_epochs, pb\_file\_path):  init = tf.global\_variables\_initializer()  with tf.Session() as sess:  sess.run(init)  epoch\_delta = 2  for epoch\_index in range(num\_epochs):  for i in range(12):  sess.run([graph['optimize']], feed\_dict={  graph['x']: np.reshape(x\_train[i], (1, 224, 224, 3)),  graph['y']: ([[1, 0]] if y\_train[i] == 0 else [[0, 1]])  }) |

其实训练的代码就这些，定好了batchsize和numepoch进行训练。下面的代码主要是为了看每几次相应的正确率。

|  |
| --- |
| constant\_graph = graph\_util.convert\_variables\_to\_constants(sess, sess.graph\_def, ["output"])  with tf.gfile.FastGFile(pb\_file\_path, mode='wb') as f:  f.write(constant\_graph.SerializeToString()) |

这两句是重要的代码，用来把训练好的模型保存为pb文件。运行完之后就会发现应该的文件夹多出了一个pb文件。

1. test

|  |
| --- |
| def recognize(jpg\_path, pb\_file\_path):  with tf.Graph().as\_default():  output\_graph\_def = tf.GraphDef()  with open(pb\_file\_path, "rb") as f:  output\_graph\_def.ParseFromString(f.read())  \_ = tf.import\_graph\_def(output\_graph\_def, name="") |

打开相应的pb文件。

|  |
| --- |
| img = io.imread(jpg\_path)  img = transform.resize(img, (224, 224, 3))  img\_out\_softmax = sess.run(out\_softmax, feed\_dict={input\_x:np.reshape(img, [-1, 224, 224, 3])}) |

读取图片文件，resize之后放入模型的输入位置，之后img\_out\_softmax就是相应输出的结果。

这大概就是整个流程。目的是为了练练手，PO主应该有挺多小错误，希望大家能够提出来让PO主好好学习哈哈哈！！！

最后放出整个的train和test的代码：   
train

|  |
| --- |
| from PIL import Image  import numpy as np  import matplotlib.pyplot as plt  import matplotlib.image as mpimg  import tensorflow as tf  import os  import glob  from skimage import io, transform  from tensorflow.python.framework import graph\_util  import collections  path = '/home/zhoupeilin/vgg16/picture/'  w = 224  h = 224  c = 3  def read\_img(path):  cate = [path + x for x in os.listdir(path) if os.path.isdir(path + x)]  imgs = []  labels = []  for idx, folder in enumerate(cate):  for im in glob.glob(folder + '/\*.jpg'):  print('reading the image: %s' % (im))  img = io.imread(im)  img = transform.resize(img, (w, h, c))  imgs.append(img)  labels.append(idx)  return np.asarray(imgs, np.float32), np.asarray(labels, np.int32)  data, label = read\_img(path)  num\_example = data.shape[0]  arr = np.arange(num\_example)  np.random.shuffle(arr)  data = data[arr]  label = label[arr]  ratio = 0.8  s = np.int(num\_example \* ratio)  x\_train = data[:s]  y\_train = label[:s]  x\_val = data[s:]  y\_val = label[s:]  def build\_network(height, width, channel):  x = tf.placeholder(tf.float32, shape=[None, height, width, channel], name='input')  y = tf.placeholder(tf.int64, shape=[None, 2], name='labels\_placeholder')  def weight\_variable(shape, name="weights"):  initial = tf.truncated\_normal(shape, dtype=tf.float32, stddev=0.1)  return tf.Variable(initial, name=name)  def bias\_variable(shape, name="biases"):  initial = tf.constant(0.1, dtype=tf.float32, shape=shape)  return tf.Variable(initial, name=name)  def conv2d(input, w):  return tf.nn.conv2d(input, w, [1, 1, 1, 1], padding='SAME')  def pool\_max(input):  return tf.nn.max\_pool(input,  ksize=[1, 2, 2, 1],  strides=[1, 2, 2, 1],  padding='SAME',  name='pool1')  def fc(input, w, b):  return tf.matmul(input, w) + b  # conv1  with tf.name\_scope('conv1\_1') as scope:  kernel = weight\_variable([3, 3, 3, 64])  biases = bias\_variable([64])  output\_conv1\_1 = tf.nn.relu(conv2d(x, kernel) + biases, name=scope)  with tf.name\_scope('conv1\_2') as scope:  kernel = weight\_variable([3, 3, 64, 64])  biases = bias\_variable([64])  output\_conv1\_2 = tf.nn.relu(conv2d(output\_conv1\_1, kernel) + biases, name=scope)  pool1 = pool\_max(output\_conv1\_2)  # conv2  with tf.name\_scope('conv2\_1') as scope:  kernel = weight\_variable([3, 3, 64, 128])  biases = bias\_variable([128])  output\_conv2\_1 = tf.nn.relu(conv2d(pool1, kernel) + biases, name=scope)  with tf.name\_scope('conv2\_2') as scope:  kernel = weight\_variable([3, 3, 128, 128])  biases = bias\_variable([128])  output\_conv2\_2 = tf.nn.relu(conv2d(output\_conv2\_1, kernel) + biases, name=scope)  pool2 = pool\_max(output\_conv2\_2)  # conv3  with tf.name\_scope('conv3\_1') as scope:  kernel = weight\_variable([3, 3, 128, 256])  biases = bias\_variable([256])  output\_conv3\_1 = tf.nn.relu(conv2d(pool2, kernel) + biases, name=scope)  with tf.name\_scope('conv3\_2') as scope:  kernel = weight\_variable([3, 3, 256, 256])  biases = bias\_variable([256])  output\_conv3\_2 = tf.nn.relu(conv2d(output\_conv3\_1, kernel) + biases, name=scope)  with tf.name\_scope('conv3\_3') as scope:  kernel = weight\_variable([3, 3, 256, 256])  biases = bias\_variable([256])  output\_conv3\_3 = tf.nn.relu(conv2d(output\_conv3\_2, kernel) + biases, name=scope)  pool3 = pool\_max(output\_conv3\_3)  # conv4  with tf.name\_scope('conv4\_1') as scope:  kernel = weight\_variable([3, 3, 256, 512])  biases = bias\_variable([512])  output\_conv4\_1 = tf.nn.relu(conv2d(pool3, kernel) + biases, name=scope)  with tf.name\_scope('conv4\_2') as scope:  kernel = weight\_variable([3, 3, 512, 512])  biases = bias\_variable([512])  output\_conv4\_2 = tf.nn.relu(conv2d(output\_conv4\_1, kernel) + biases, name=scope)  with tf.name\_scope('conv4\_3') as scope:  kernel = weight\_variable([3, 3, 512, 512])  biases = bias\_variable([512])  output\_conv4\_3 = tf.nn.relu(conv2d(output\_conv4\_2, kernel) + biases, name=scope)  pool4 = pool\_max(output\_conv4\_3)  # conv5  with tf.name\_scope('conv5\_1') as scope:  kernel = weight\_variable([3, 3, 512, 512])  biases = bias\_variable([512])  output\_conv5\_1 = tf.nn.relu(conv2d(pool4, kernel) + biases, name=scope)  with tf.name\_scope('conv5\_2') as scope:  kernel = weight\_variable([3, 3, 512, 512])  biases = bias\_variable([512])  output\_conv5\_2 = tf.nn.relu(conv2d(output\_conv5\_1, kernel) + biases, name=scope)  with tf.name\_scope('conv5\_3') as scope:  kernel = weight\_variable([3, 3, 512, 512])  biases = bias\_variable([512])  output\_conv5\_3 = tf.nn.relu(conv2d(output\_conv5\_2, kernel) + biases, name=scope)  pool5 = pool\_max(output\_conv5\_3)  #fc6  with tf.name\_scope('fc6') as scope:  shape = int(np.prod(pool5.get\_shape()[1:]))  kernel = weight\_variable([shape, 4096])  biases = bias\_variable([4096])  pool5\_flat = tf.reshape(pool5, [-1, shape])  output\_fc6 = tf.nn.relu(fc(pool5\_flat, kernel, biases), name=scope)  #fc7  with tf.name\_scope('fc7') as scope:  kernel = weight\_variable([4096, 4096])  biases = bias\_variable([4096])  output\_fc7 = tf.nn.relu(fc(output\_fc6, kernel, biases), name=scope)  #fc8  with tf.name\_scope('fc8') as scope:  kernel = weight\_variable([4096, 2])  biases = bias\_variable([2])  output\_fc8 = tf.nn.relu(fc(output\_fc7, kernel, biases), name=scope)  finaloutput = tf.nn.softmax(output\_fc8, name="softmax")  cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=finaloutput, labels=y))  optimize = tf.train.AdamOptimizer(learning\_rate=1e-4).minimize(cost)  prediction\_labels = tf.argmax(finaloutput, axis=1, name="output")  read\_labels = y  correct\_prediction = tf.equal(prediction\_labels, read\_labels)  accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  correct\_times\_in\_batch = tf.reduce\_sum(tf.cast(correct\_prediction, tf.int32))  return dict(  x=x,  y=y,  optimize=optimize,  correct\_prediction=correct\_prediction,  correct\_times\_in\_batch=correct\_times\_in\_batch,  cost=cost,  )  def train\_network(graph, batch\_size, num\_epochs, pb\_file\_path):  init = tf.global\_variables\_initializer()  with tf.Session() as sess:  sess.run(init)  epoch\_delta = 2  for epoch\_index in range(num\_epochs):  for i in range(12):  sess.run([graph['optimize']], feed\_dict={  graph['x']: np.reshape(x\_train[i], (1, 224, 224, 3)),  graph['y']: ([[1, 0]] if y\_train[i] == 0 else [[0, 1]])  })  if epoch\_index % epoch\_delta == 0:  total\_batches\_in\_train\_set = 0  total\_correct\_times\_in\_train\_set = 0  total\_cost\_in\_train\_set = 0.  for i in range(12):  return\_correct\_times\_in\_batch = sess.run(graph['correct\_times\_in\_batch'], feed\_dict={  graph['x']: np.reshape(x\_train[i], (1, 224, 224, 3)),  graph['y']: ([[1, 0]] if y\_train[i] == 0 else [[0, 1]])  })  mean\_cost\_in\_batch = sess.run(graph['cost'], feed\_dict={  graph['x']: np.reshape(x\_train[i], (1, 224, 224, 3)),  graph['y']: ([[1, 0]] if y\_train[i] == 0 else [[0, 1]])  })  total\_batches\_in\_train\_set += 1  total\_correct\_times\_in\_train\_set += return\_correct\_times\_in\_batch  total\_cost\_in\_train\_set += (mean\_cost\_in\_batch \* batch\_size)  total\_batches\_in\_test\_set = 0  total\_correct\_times\_in\_test\_set = 0  total\_cost\_in\_test\_set = 0.  for i in range(3):  return\_correct\_times\_in\_batch = sess.run(graph['correct\_times\_in\_batch'], feed\_dict={  graph['x']: np.reshape(x\_val[i], (1, 224, 224, 3)),  graph['y']: ([[1, 0]] if y\_val[i] == 0 else [[0, 1]])  })  mean\_cost\_in\_batch = sess.run(graph['cost'], feed\_dict={  graph['x']: np.reshape(x\_val[i], (1, 224, 224, 3)),  graph['y']: ([[1, 0]] if y\_val[i] == 0 else [[0, 1]])  })  total\_batches\_in\_test\_set += 1  total\_correct\_times\_in\_test\_set += return\_correct\_times\_in\_batch  total\_cost\_in\_test\_set += (mean\_cost\_in\_batch \* batch\_size)  acy\_on\_test = total\_correct\_times\_in\_test\_set / float(total\_batches\_in\_test\_set \* batch\_size)  acy\_on\_train = total\_correct\_times\_in\_train\_set / float(total\_batches\_in\_train\_set \* batch\_size)  print('Epoch - {:2d}, acy\_on\_test:{:6.2f}%({}/{}),loss\_on\_test:{:6.2f}, acy\_on\_train:{:6.2f}%({}/{}),loss\_on\_train:{:6.2f}'.format(epoch\_index, acy\_on\_test\*100.0,total\_correct\_times\_in\_test\_set,  total\_batches\_in\_test\_set \* batch\_size,  total\_cost\_in\_test\_set,  acy\_on\_train \* 100.0,  total\_correct\_times\_in\_train\_set,  total\_batches\_in\_train\_set \* batch\_size,  total\_cost\_in\_train\_set))  constant\_graph = graph\_util.convert\_variables\_to\_constants(sess, sess.graph\_def, ["output"])  with tf.gfile.FastGFile(pb\_file\_path, mode='wb') as f:  f.write(constant\_graph.SerializeToString())  def main():  batch\_size = 12  num\_epochs = 50  pb\_file\_path = "vggs.pb"  g = build\_network(height=224, width=224, channel=3)  train\_network(g, batch\_size, num\_epochs, pb\_file\_path)  main() |

test

|  |
| --- |
| import tensorflow as tf  import numpy as np  import PIL.Image as Image  from skimage import io, transform  def recognize(jpg\_path, pb\_file\_path):  with tf.Graph().as\_default():  output\_graph\_def = tf.GraphDef()  with open(pb\_file\_path, "rb") as f:  output\_graph\_def.ParseFromString(f.read())  \_ = tf.import\_graph\_def(output\_graph\_def, name="")  with tf.Session() as sess:  init = tf.global\_variables\_initializer()  sess.run(init)  input\_x = sess.graph.get\_tensor\_by\_name("input:0")  print input\_x  out\_softmax = sess.graph.get\_tensor\_by\_name("softmax:0")  print out\_softmax  out\_label = sess.graph.get\_tensor\_by\_name("output:0")  print out\_label  img = io.imread(jpg\_path)  img = transform.resize(img, (224, 224, 3))  img\_out\_softmax = sess.run(out\_softmax, feed\_dict={input\_x:np.reshape(img, [-1, 224, 224, 3])})  print "img\_out\_softmax:",img\_out\_softmax  prediction\_labels = np.argmax(img\_out\_softmax, axis=1)  print "label:",prediction\_labels  recognize("vgg16/picture/dog/dog3.jpg", "vgg16/vggs.pb") |