#打开一个图像

im=Image.open(os.path.join(r"C:\Users\yy\\DatasetA\_test\test16", filename ), 'r')

label\_arr = np.array([filename[2]])

# 将图像的RGB分离  
r, g, b = im.split()

# 将PILLOW图像转成数组

r\_arr = plimg.pil\_to\_array(r)

g\_arr=......

#将数组转化为一维格式  
r\_arr1 = r\_arr.reshape(size\*size)

#将三个通道的数组合并  
arr = np.concatenate((r\_arr1, g\_arr1, b\_arr1))

#归一化处理

def normalize(x):  
 a = 0  
 b = 1  
 grayscale\_min = np.min(x)  
 grayscale\_max = np.max(x)  
  **t = a + ( (x - grayscale\_min)\*(b - a) )/( grayscale\_max - grayscale\_min )**  
 return t

#对标签进行编码（one-hot）

def one\_hot\_encode(x):  
 x = np.array(x)   
 #print (x)  
 num\_labels = x.shape[0]  
 x\_one\_hot = np.zeros((num\_labels,6))  
 for i in np.arange(num\_labels):  
 x\_one\_hot[i][x[i]] = 1  
 return x\_one\_hot

#将标准化后的图片数组转化为原图片的形状

features = np.array(batch['data']).reshape((len(batch['data']), 3, size, size)).transpose(0, 2, 3, 1)

# 卷积和最大池化层  
def conv2d\_maxpool(x\_tensor, conv\_num\_outputs, conv\_ksize, conv\_strides, pool\_ksize, pool\_strides):  
 w = tf.Variable(  
 tf.truncated\_normal([conv\_ksize[0], conv\_ksize[1], x\_tensor.get\_shape().as\_list()[3], conv\_num\_outputs],stddev=0.05)**)#初始化卷积核为正态分布的随机数**  
 b = tf.Variable(tf.truncated\_normal([conv\_num\_outputs], stddev=0.05))  
 **x = tf.nn.conv2d(x\_tensor, w, [1, conv\_strides[0], conv\_strides[1], 1], padding='SAME')**  
 x = tf.nn.bias\_add(x, b)  
 **x = tf.nn.relu(x)**  
 **x = tf.nn.max\_pool(x, [1, pool\_ksize[0], pool\_ksize[1], 1],**

**[1, pool\_strides[0], pool\_strides[1], 1],padding="SAME")**  
 return x

# 扁平化层  
def flatten(x\_tensor):  
 x\_shape = x\_tensor.get\_shape().as\_list()  
 x\_tensor = tf.reshape(x\_tensor, shape=[-1, x\_shape[1] \* x\_shape[2] \* x\_shape[3]])  
 return x\_tensor

# 全连接层  
  
def fully\_conn(x\_tensor, num\_outputs):  
  
 batch, size = x\_tensor.get\_shape().as\_list()  
 w = tf.Variable(tf.truncated\_normal([size, num\_outputs], stddev=0.05))  
 b = tf.Variable(tf.truncated\_normal([num\_outputs], stddev=0.05))  
 fc1 = tf.matmul(x\_tensor, w)  
 fc1 = tf.add(fc1, b)  
 **fc1 = tf.nn.relu(fc1)**  
 return fc1

# 输出层

def output(x\_tensor, num\_outputs):  
  
 batch, size = x\_tensor.get\_shape().as\_list()  
 w = tf.Variable(tf.truncated\_normal([size, num\_outputs], stddev=0.05))  
 b = tf.Variable(tf.truncated\_normal([num\_outputs], stddev=0.05))  
 fc1 = tf.add(tf.matmul(x\_tensor, w), b)  
 return fc1

# 构建网络  
def conv\_net(x, keep\_prob):  
 # TODO: Apply 1, 2, or 3 Convolution and Max Pool layers  
 # Play around with different number of outputs, kernel size and stride  
 # Function Definition from Above:  
 # conv\_num\_outputs:滤波器的种类个数  
 # conv2d\_maxpool(x\_tensor, conv\_num\_outputs, conv\_ksize, conv\_strides, pool\_ksize, pool\_strides)  
 **con = conv2d\_maxpool(x, 32, (5, 5), (2, 2), (2, 2), (2, 2))  
 con = conv2d\_maxpool(con, 64, (3, 3), (1, 1), (2, 2), (2, 2))  
 con = conv2d\_maxpool(con, 128, (2, 2), (1, 1), (2, 2), (2, 2))**  
 con = flatten(con)  
  
 fc = fully\_conn(con, 512) # 1024  
 **fc = tf.nn.dropout(fc, keep\_prob)**  
 fc = fully\_conn(fc, 128) # 512  
 fc = tf.nn.dropout(fc, keep\_prob)  
 fc = fully\_conn(fc, 32)  
 fc = tf.nn.dropout(fc, keep\_prob)  
  
 out = output(fc, 6)  
  
 return out

#dropout神经元随机失活

epochs = 300 # 训练周期数  
batch\_size = 224  
keep\_probability = 0.75

learning\_rate = tf.train.exponential\_decay(learning\_rate\_start, global\_, decay\_steps, decay\_rate, staircase=True)

# Loss and Optimizer  
cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logits, labels=y))  
# optimizer = tf.train.AdamOptimizer().minimize(cost)  
optimizer = tf.train.GradientDescentOptimizer(learning\_rate).minimize(cost,global\_step = global\_)  
# Accuracy  
correct\_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1)) # tf.argmax获取logits中的最大值的索引  
# correct\_pred = tf.equal(logits,y)  
# jin = tf.cast(correct\_pred, tf.float32)  
accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32), name='accuracy') # tf.cast将true变成float(1/0)

# 训练神经网络

session.run(optimizer, feed\_dict={x: feature\_batch, y: label\_batch, keep\_prob: keep\_probability})