# 修改AlexNet实现验证码识别

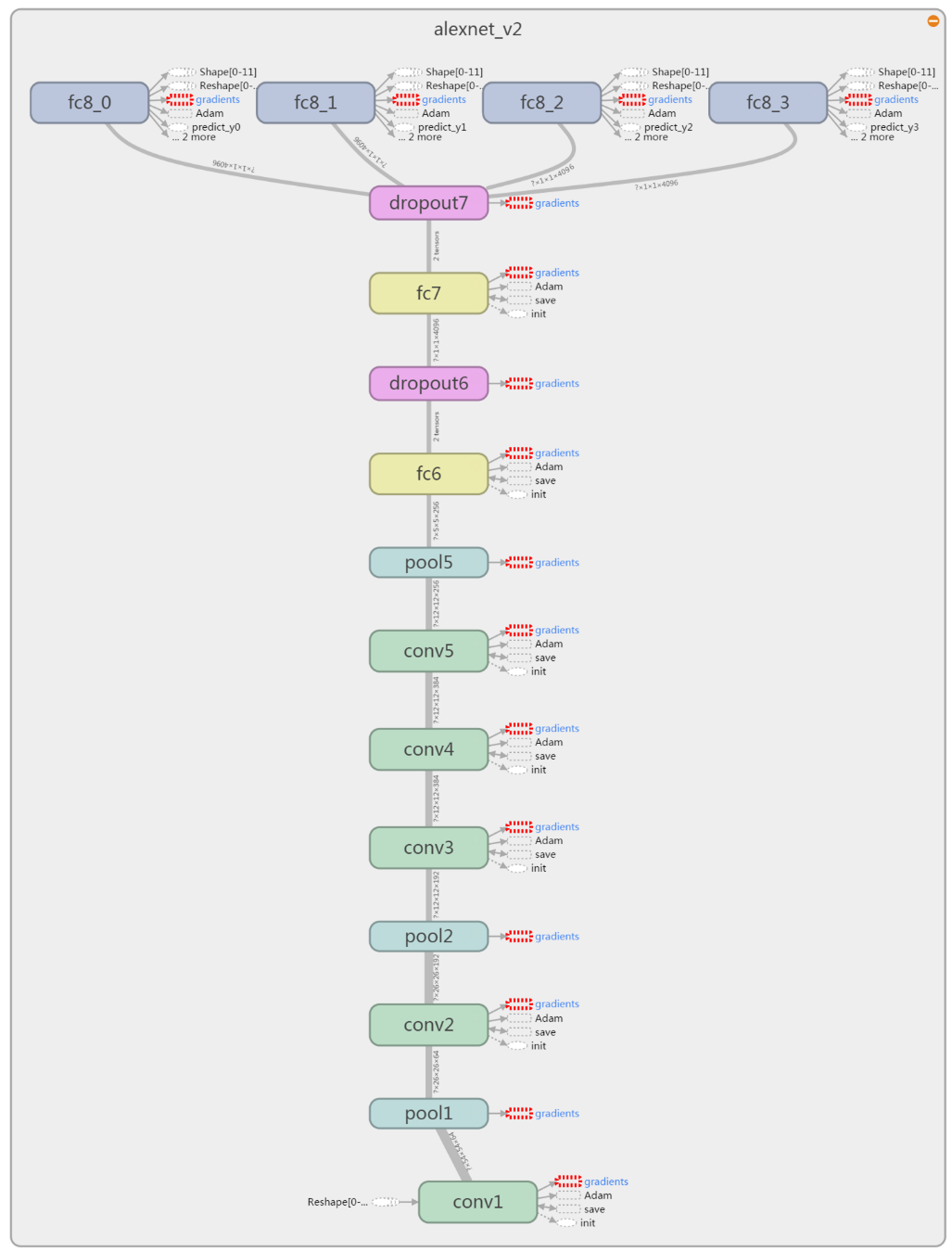
## 原理

将原先一个输出改成4个输出，相当于4个网络，除最后一层外，其它层共用。

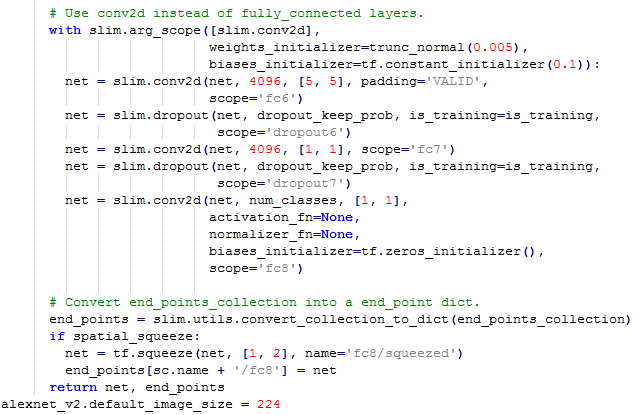
## 2. 实现

### 2.1修改网络结构

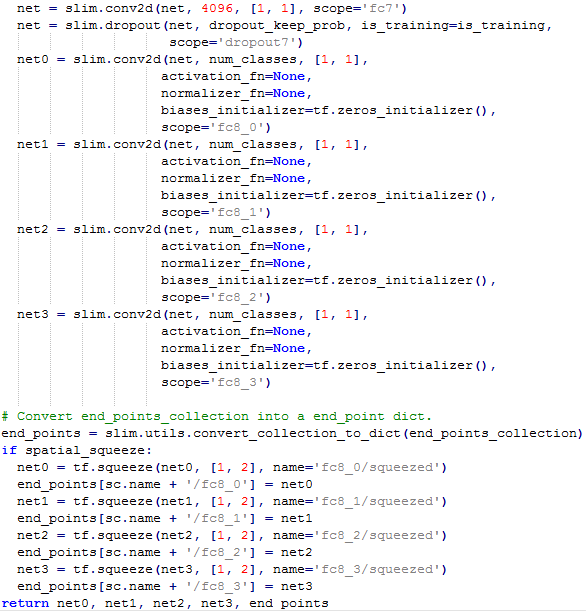
AlexNet网络原只有一个输出，为适应多任务识别，将其改成多个输出，网络结构如下：

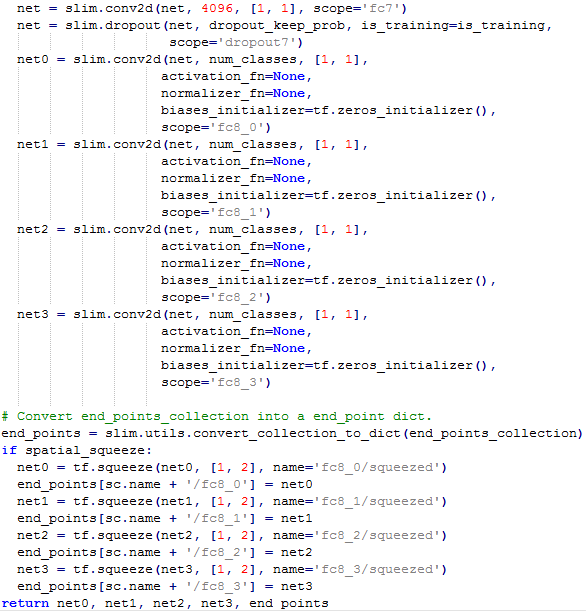


在github上下载models仓库，地址：<https://github.com/tensorflow/models>，在其nets文件夹下实现了很多经典网络，打开alexnet.py文件，后半部分代码如下：



将其改成：





将nets目录拷到工程目录下。

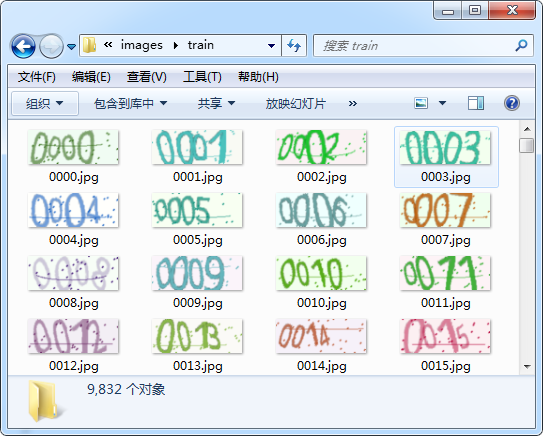
### 2.2 数据准备

#### 1) 生成验证码

生成验证码不是本文的重点，代码如下：

|  |
| --- |
| *# coding: utf-8 # 验证码生成库* **from** captcha.image **import** ImageCaptcha *# pip install captcha* **import** random **import** sys   number = [**'0'**,**'1'**,**'2'**,**'3'**,**'4'**,**'5'**,**'6'**,**'7'**,**'8'**,**'9'**] i = 0 num = 20000  **def** random\_captcha\_text(char\_set=number, captcha\_size=4):  *# 验证码列表* captcha\_text = []  **for** i **in** range(captcha\_size):  *#随机选择* c = random.choice(char\_set)  *#加入验证码列表* captcha\_text.append(c)  **return** captcha\_text  **def** gen\_captcha\_text\_and\_image(index):  image = ImageCaptcha()  *#获得随机生成的验证码* captcha\_text = random\_captcha\_text()  *#把验证码列表转为字符串* captcha\_text = **''**.join(captcha\_text)  *#生成验证码* captcha = image.generate(captcha\_text)  **if** index < 0.5 \* num:  image.write(captcha\_text, **'images/train/'** + captcha\_text + **'.jpg'**) *# 写到文件* **else**:  image.write(captcha\_text, **'images/val/'** + captcha\_text + **'.jpg'**) *# 写到文件* **if** \_\_name\_\_ == **'\_\_main\_\_'**:  **for** i **in** range(num):  gen\_captcha\_text\_and\_image(i)  sys.stdout.write(**'\r>> Creating image %d/%d\n'** % (i+1, num))  sys.stdout.flush()  sys.stdout.write(**'\n'**)  sys.stdout.flush()  print(**"生成完毕"**) |

生成验证码：



#### 2) 生成tfrecords文件

根据文件名生成对应的标签，4个标签相互独立，以一个标题数字返回，而不是以一个4个元素的向量。

首先定义一个TFRecordWriter，使用PIL的Image.open函数打开图像文件，缩放到[224, 224]大小，使用img.convert(**'L'**)转成灰度图像，并转成numpy的array类型，最后转成byte类型。

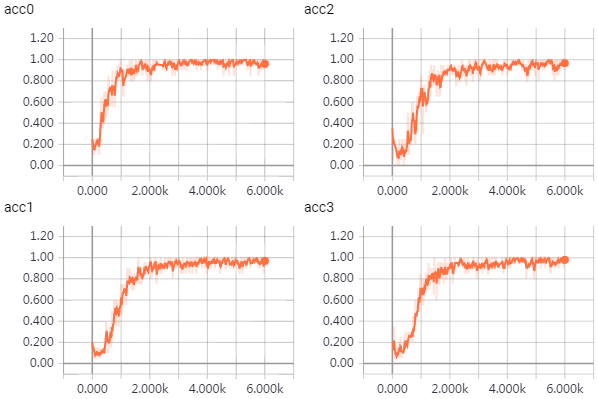
将图像与标签作为特征，打包到example中，序列化到文件中。

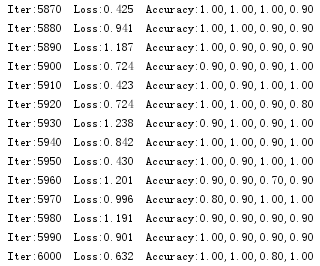
|  |
| --- |
| *# coding: utf-8* **import** tensorflow **as** tf **import** os **from** PIL **import** Image **import** numpy **as** np  **def** get\_labels\_from\_image\_name(image\_name):  **return** int(image\_name[0]), int(image\_name[1]), int(image\_name[2]), int(image\_name[3])  **def** createTfrecord(folder, tfrecord\_file):  writer = tf.python\_io.TFRecordWriter(tfrecord\_file)  print(**'convert start ...'**)  **for** img\_name **in** os.listdir(folder):  img\_path = os.path.join(folder, img\_name)  **try**:  label\_0, label\_1, label\_2, label\_3 = get\_labels\_from\_image\_name(img\_name)  img = Image.open(img\_path)  img = img.resize([224, 224])  img = np.array(img.convert(**'L'**))  img = img.tobytes()  example = tf.train.Example(features=tf.train.Features(feature={  **'image'**: tf.train.Feature(bytes\_list=tf.train.BytesList(value=[img])),  **"label\_0"**: tf.train.Feature(int64\_list=tf.train.Int64List(value=[label\_0])),  **"label\_1"**: tf.train.Feature(int64\_list=tf.train.Int64List(value=[label\_1])),  **"label\_2"**: tf.train.Feature(int64\_list=tf.train.Int64List(value=[label\_2])),  **"label\_3"**: tf.train.Feature(int64\_list=tf.train.Int64List(value=[label\_3]))  }))  writer.write(example.SerializeToString())  **except** IOError **as** e:  print(**'cannot open '**, img\_path)  writer.close()  print(**'convert done!'**)  createTfrecord(**'images/train'**, **'train.tfrecords'**) |

### 2.3 训练模型

在训练过程中，将模型保存以方便在测试时将模型恢复。

|  |
| --- |
| *#coding=utf-8* **import** tensorflow **as** tf **from** nets **import** nets\_factory  TFRECORD\_FILE = **'train.tfrecords'** BATCH\_SIZE = 20 CLASS\_NUM = 10  *#输入* input\_x = tf.placeholder(tf.float32, [**None**, 224, 224], name = **'input\_x'**) input\_y0 = tf.placeholder(tf.float32, [**None**], name = **'input\_y0'**) input\_y1 = tf.placeholder(tf.float32, [**None**], name = **'input\_y1'**) input\_y2 = tf.placeholder(tf.float32, [**None**], name = **'input\_y2'**) input\_y3 = tf.placeholder(tf.float32, [**None**], name = **'input\_y3'**)  *#学习率* lr = tf.Variable(0.003, dtype=tf.float32) *#从tfrecord读出数据* **def** read\_and\_decode(tfrecords\_file):  *#创建队列* filename\_queue = tf.train.string\_input\_producer([tfrecords\_file])  *#读取文件* reader = tf.TFRecordReader()  \_, serialized\_example = reader.read(filename\_queue)  *#导出特征* features = tf.parse\_single\_example(serialized\_example,  features={  **'image'**: tf.FixedLenFeature([], tf.string),  **'label\_0'**: tf.FixedLenFeature([], tf.int64),  **'label\_1'**: tf.FixedLenFeature([], tf.int64),  **'label\_2'**: tf.FixedLenFeature([], tf.int64),  **'label\_3'**: tf.FixedLenFeature([], tf.int64)  })  *#获取图片数据* image = tf.decode\_raw(features[**'image'**], tf.uint8)  image = tf.reshape(image, [224, 224])  *#预处理* image = tf.cast(image, tf.float32) / 255.0  image = tf.subtract(image, 0.5)  image = tf.multiply(image, 2.0)  *#获取label* label\_0 = tf.cast(features[**'label\_0'**], tf.int32)  label\_1 = tf.cast(features[**'label\_1'**], tf.int32)  label\_2 = tf.cast(features[**'label\_2'**], tf.int32)  label\_3 = tf.cast(features[**'label\_3'**], tf.int32)  **return** image, label\_0, label\_1, label\_2, label\_3 *#读取图像与标签* image, label\_0, label\_1, label\_2, label\_3 = read\_and\_decode(**'train.tfrecords'**) *#生成Batch* image\_batch, label\_0\_batch,label\_1\_batch, label\_2\_batch, label\_3\_batch = tf.train.shuffle\_batch(  [image, label\_0, label\_1, label\_2, label\_3], batch\_size = BATCH\_SIZE,  capacity=50000, min\_after\_dequeue=10000, num\_threads=1 ) *#定义网络* net\_fn = nets\_factory.get\_network\_fn(  name=**'alexnet\_v2'**, num\_classes=CLASS\_NUM,weight\_decay=0.0005,is\_training=**True** )  **with** tf.Session() **as** sess:  input\_X = tf.reshape(input\_x, [-1, 224, 224, 1])  logist0, logist1, logist2, logist3, end\_points = net\_fn(input\_X)  one\_hot\_label\_0 = tf.one\_hot(indices = tf.cast(input\_y0, tf.int32), depth = CLASS\_NUM)  one\_hot\_label\_1 = tf.one\_hot(indices = tf.cast(input\_y1, tf.int32), depth = CLASS\_NUM)  one\_hot\_label\_2 = tf.one\_hot(indices = tf.cast(input\_y2, tf.int32), depth = CLASS\_NUM)  one\_hot\_label\_3 = tf.one\_hot(indices = tf.cast(input\_y3, tf.int32), depth = CLASS\_NUM)  *#计算loss* loss\_0 = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logist0, labels=one\_hot\_label\_0))  loss\_1 = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logist1, labels=one\_hot\_label\_1))  loss\_2 = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logist2, labels=one\_hot\_label\_2))  loss\_3 = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=logist3, labels=one\_hot\_label\_3))  total\_loss = (loss\_0 + loss\_1 + loss\_2 + loss\_3)  *#优化* optimzer = tf.train.AdamOptimizer(learning\_rate=lr).minimize(total\_loss)  *#准确率* predict\_y0 = tf.argmax(logist0, 1, name = **'predict\_y0'**)  predict\_y1 = tf.argmax(logist1, 1, name = **'predict\_y1'**)  predict\_y2 = tf.argmax(logist2, 1, name = **'predict\_y2'**)  predict\_y3 = tf.argmax(logist3, 1, name = **'predict\_y3'**)  correct\_prediction\_0 = tf.equal(tf.argmax(one\_hot\_label\_0, 1), predict\_y0)  accuracy\_0 = tf.reduce\_mean(tf.cast(correct\_prediction\_0, tf.float32))  correct\_prediction\_1 = tf.equal(tf.argmax(one\_hot\_label\_1, 1), predict\_y1)  accuracy\_1 = tf.reduce\_mean(tf.cast(correct\_prediction\_1, tf.float32))  correct\_prediction\_2 = tf.equal(tf.argmax(one\_hot\_label\_2, 1), predict\_y2)  accuracy\_2 = tf.reduce\_mean(tf.cast(correct\_prediction\_2, tf.float32))  correct\_prediction\_3 = tf.equal(tf.argmax(one\_hot\_label\_3, 1), predict\_y3)  accuracy\_3 = tf.reduce\_mean(tf.cast(correct\_prediction\_3, tf.float32))  tf.summary.scalar(**'acc0'**, accuracy\_0)  tf.summary.scalar(**'acc1'**, accuracy\_1)  tf.summary.scalar(**'acc2'**, accuracy\_2)  tf.summary.scalar(**'acc3'**, accuracy\_3)  merged = tf.summary.merge\_all()  *#保存模型* saver = tf.train.Saver()  *#初始化* sess.run(tf.global\_variables\_initializer())   coord = tf.train.Coordinator()  threads = tf.train.start\_queue\_runners(sess = sess, coord = coord)  writer = tf.summary.FileWriter(**'logs/'**, sess.graph)  **for** i **in** range(6001):  b\_image, b\_label\_0, b\_label\_1, b\_label\_2, b\_label\_3 = sess.run([image\_batch, label\_0\_batch,label\_1\_batch,  label\_2\_batch, label\_3\_batch])  dict = {input\_x:b\_image, input\_y0:b\_label\_0,input\_y1:b\_label\_1,input\_y2:b\_label\_2, input\_y3:b\_label\_3}  sess.run(optimzer, feed\_dict = dict)  **if** i % 20 == 0:  **if** i%6000 == 0:  sess.run(tf.assign(lr, lr/3))  saver.save(sess, **"models/captcha\_model.ckpt"**, global\_step = i)  acc0, acc1, acc2, acc3, loss = sess.run([accuracy\_0, accuracy\_1, accuracy\_2, accuracy\_3, total\_loss], feed\_dict = dict)  summary = sess.run(merged, feed\_dict=dict)  writer.add\_summary(summary, i)  print(**"Iter:%d Loss:%.3f Accuracy:%.2f,%.2f,%.2f,%.2f"** % (i, loss, acc0, acc1, acc2, acc3))  coord.request\_stop() coord.join(threads) |

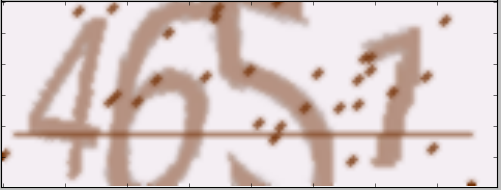


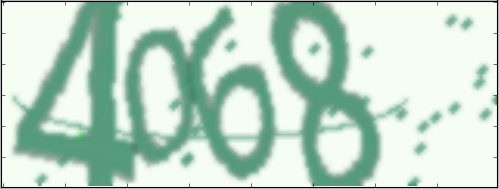


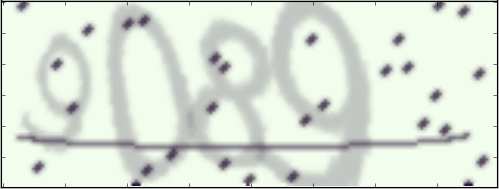
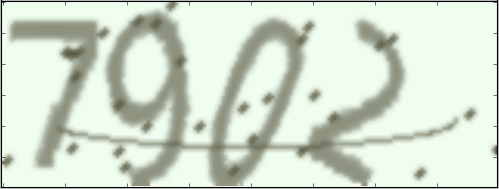
### 2.4 测试模型

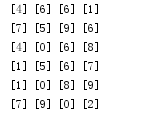
模型的恢复，使用**captcha\_model.ckpt-6000.meta文件**恢复网络结构，使用**models/captcha\_model.ckpt-6000**文件来恢复模型参数，恢复方法及步骤参考以下程序。需要注意的是，必须恢复输入的placeholder和预测结果tensor，在测试模型时，只要将数据喂给输入placeholder即可从预测结构tensor中得到预测结果。喂给placeholder的必须是真正的数据，而不是tensor，因为读取图像得到的image是一个tensor，需要使用image.eval()来得到其数据。以下演示了直接输入图像格式的数据，得到预测结果的方法。

|  |
| --- |
| *#coding=utf-8* **import** tensorflow **as** tf **from** PIL **import** Image **import** matplotlib.pyplot **as** plt **import** numpy **as** np **import** random **import** os  **def** get\_one\_image(image\_path):  image = Image.open(image\_path)  plt.imshow(image)  image = image.resize([224,224])  image = np.array(image.convert(**'L'**))  image = tf.cast(image, tf.float32) / 255.0  image = tf.subtract(image, 0.5)  image = tf.multiply(image, 2.0)  **return** image  saver = tf.train.import\_meta\_graph(**'models/captcha\_model.ckpt-6000.meta'**) **with** tf.Session() **as** sess:  saver.restore(sess, **'models/captcha\_model.ckpt-6000'**)  graph = tf.get\_default\_graph()  input\_x = graph.get\_tensor\_by\_name(**'input\_x:0'**)  pred\_y0 = graph.get\_tensor\_by\_name(**'predict\_y0:0'**)  pred\_y1 = graph.get\_tensor\_by\_name(**'predict\_y1:0'**)  pred\_y2 = graph.get\_tensor\_by\_name(**'predict\_y2:0'**)  pred\_y3 = graph.get\_tensor\_by\_name(**'predict\_y3:0'**)  filelist = [os.path.join(**'images/val'**, filename) **for** filename **in** os.listdir(**'images/val'**)]  random.shuffle(filelist)  **for** image\_path **in** filelist:  image = get\_one\_image(image\_path)  image = tf.reshape(image, [1, 224, 224])  y0, y1, y2, y3 = sess.run([pred\_y0, pred\_y1, pred\_y2, pred\_y3], feed\_dict={input\_x: image.eval()})  print(y0 ,y1, y2, y3)  plt.show() |



## 总结

这里演示了修改AlexNet来实现验证码识别，内容比较丰富，如slim中nets的使用，tfrecords文件的创建，数据输入pipeline，模型的保存与恢复，变量的监控，图像预处理，单张图像的测试，代码也较为全面，可供其它案例参考。