Tensorflow 笔记：第八讲

本节目标：复现已有网络，实现实际应用。

一、VGG 论文阅读笔记

笔记阅读2015年发表于ICLR的《VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION》图像分类部分，课程实践采用的是VGG-16模型。

摘要

论文探讨在大规模数据集情景下，卷积网络深度对其准确率的影响。我们的主要贡献在于利用3*3小卷积核的网络结构对逐渐加深的网络进行全面的评估，结果表明加深网络深度到16至19层可极大超越前人的网络结构。这些成果是基于2014年的ImageNet挑战赛，该模型在定位和分类跟踪比赛中分别取得了第一名和第二名。同时模型在其他数据集上也有较好的泛化性。我们公开了两个表现最好的卷积神经网络模型，以促进计算机视觉领域模型的进一步研究。

1. 介绍

卷积网络(ConvNets)成功原因：大型公共图像数据集，如 ImageNet；高性能计算系统，如GPU或大规模分布式集群。

前人的工作：人们尝试在AlexNet的原始框架上做一些改进。比如在第一个卷积上使用较小的卷积核以及较小的滑动步长。另一种方法则是在全图以及多个尺寸上，稠密的训练和测试网络。

而本文主要关注网络的深度。为此，我们固定网络的其他参数，通过增加卷积层来增加网络深度，这是可行的，因为我们所有层都采用小的3*3卷积核。

2. 卷积配置

为凸显深度对模型效果的影响，我们所有卷积采用相同配置。本章先介绍卷积网络的通用架构，再描述其评估中的具体细节，最后讨论我们的设计选择以及前人网络的比较。

2.1 架构

训练输入：固定尺寸224*224的RGB图像。

预处理：每个像素值减去训练集上的RGB均值。

卷积核：一系列3*3卷积核堆叠，步长为1，采用padding保持卷积后图像空间分辨率不变。

空间池化：紧随卷积“堆”的最大池化，为2*2滑动窗口，步长为2。

全连接层：特征提取完成后，接三个全连接层，前两个为4096通道，第三个为1000通道，最后是一个soft-max层，输出概率。

所有隐藏层都用非线性修正ReLu。

2.2 详细配置

表1中每列代表不同的网络，只有深度不同（层数计算不包含池化层）。卷积的通道数量很小，第一层仅64通道，每经过一次最大池化，通道数翻倍，知道数量达到512通道。

表2展示了每种模型的参数数量，尽管网络加深，但权重并未大幅增加，因为参数量主要集中在全连接层。

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as “conv⟨receptive field size⟩-⟨number of channels⟩”. The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

2.3　讨论

两个3*3卷积核相当于一个5*5卷积核的感受域，三个3*3卷积核相当于一个7*7卷积核的感受域。

优点：三个卷积堆叠具有三个非线性修正层，使模型更具判别性；其次三个3*3卷积参数量更少，相当于在7*7卷积核上加入了正则化。

3. 分类框架

3.1训练

训练方法基本与AlexNet一致，除了多尺度训练图像采样方法不一致。

训练采用mini-batch梯度下降法，batch size=256；

采用动量优化算法，momentum=0.9；

采用L2正则化方法，惩罚系数0.00005；dropout比率设为0.5；

初始学习率为0.001，当验证集准确率不再提高时，学习率衰减为原来的0.1倍，总共下降三次；

总迭代次数为370K（74epochs）；

数据增强采用随机裁剪，水平翻转，RGB颜色变化；

设置训练图片大小的两种方法：

定义S代表经过各向同性缩放的训练图像的最小边。

第一种方法针对单尺寸图像训练，S=256或384，输入图片从中随机裁剪224*224大小的图片，原则上S可以取任意不小于224的值。

第二种方法是多尺度训练，每张图像单独从$[S_{min},S_{max}]$中随机选取S来进行尺寸缩放，由于图像中目标物体尺寸不定，因此训练中采用这种方法是有效的，可看作一种尺寸抖动的训练集数据增强。

论文中提到，网络权重的初始化非常重要，由于深度网络梯度的不稳定性，不合适的初始化会阻碍网络的学习。因此我们先训练浅层网络，再用训练好的浅层网络去初始化深层网络。

3.2测试

测试阶段，对于已训练好的卷积网络和一张输入图像，采用以下方法分类：

首先，图像的最小边被各向同性的缩放到预定尺寸Q；

然后，将原先的全连接层改换成卷积层，在未裁剪的全图像上运用卷积网络，输出是一个与输入图像尺寸相关的分类得分图，输出通道数与类别数相同；

最后，对分类得分图进行空间平均化，得到固定尺寸的分类得分向量。

我们同样对测试集做数据增强，采用水平翻转，最终取原始图像和翻转图像的soft-max分类概率的平均值作为最终得分。

由于测试阶段采用全卷积网络，无需对输入图像进行裁剪，相对于多重裁剪效率会更高。但多重裁剪评估和运用全卷积的密集评估是互补的，有助于性能提升。

4. 分类实验

4.1单尺寸评估

表3展示单一测试尺寸上的卷积网络性能

Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

4.2多尺寸评估

表4展示多个测试尺寸上的卷积网络性能

Table 4: **ConvNet performance at multiple test scales.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

4.3 多重裁剪与密集网络评估

表 5 展示多重裁剪与密集网络对比，并展示两者相融合的效果

Table 5: **ConvNet evaluation techniques comparison.** In all experiments the training scale S was sampled from $[256; 512]$, and three test scales Q were considered: $\{256, 384, 512\}$.

ConvNet config. (Table 1)	Evaluation method	top-1 val. error (%)	top-5 val. error (%)
D	dense	24.8	7.5
	multi-crop	24.6	7.5
	multi-crop & dense	**24.4**	**7.2**
E	dense	24.8	7.5
	multi-crop	24.6	7.4
	multi-crop & dense	**24.4**	**7.1**

4.4 卷积模型的融合

这部分探讨不同模型融合的性能，计算多个模型的 soft-max 分类概率的平均值来对它们的输出进行组合，由于模型的互补性，性能有所提高，这也用于比赛的最佳结果中。

表 6 展示多个卷积网络融合的结果

Table 6: **Multiple ConvNet fusion results.**

Combined ConvNet models	Error		
	top-1 val	top-5 val	top-5 test
ILSVRC submission			
(D/256/224,256,288), (D/384/352,384,416), (D/[256;512]/256,384,512) (C/256/224,256,288), (C/384/352,384,416) (E/256/224,256,288), (E/384/352,384,416)	24.7	7.5	7.3
post-submission			
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), dense eval.	24.0	7.1	7.0
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), multi-crop	23.9	7.2	-
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), multi-crop & dense eval.	**23.7**	**6.8**	**6.8**

4.5 与当前最好算法的比较

表七展示对当前最好算法的对比

Table 7: **Comparison with the state of the art in ILSVRC classification**. Our method is denoted as "VGG". Only the results obtained without outside training data are reported.

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

5．结论

本文评估了非常深的卷积网络在大规模图像分类上的性能。结果表明深度有

利于分类准确率的提升。附录中展示了模型的泛化能力，再次确认了视觉表达中深度的重要性。

二、VGG 实现代码重点讲解

√x = tf.placeholder(tf.float32,shape = [BATCH_SIZE,IMAGE_PIXELS])

tf.placeholder：用于传入真实训练样本 / 测试 / 真实特征 / 待处理特征。只是占位,不必给出初值。用 sess.run 的 feed_dict 参数以字典形式喂入 x:, y_:

sess.run(feed_dict = {x: ,y_: })

BATCH_SIZE：一次传入的个数。

IMAGE_PIXELS：图像像素。

例：x = tf.placeholder("float",[1,224,224,3])

BATCH_SIZE 为 1，表示一次传入一个。图像像素为[224,224,3]。

√w = tf.Variable(tf.random_normal())：从正态分布中给出权重 w 的随机值。

b = tf.Variable(tf.zeros())：统一将偏置 b 初始化为 0。

注意：以上两行函数 Variable 中的 V 要大写，Variable 必须给初值。

√np.load　np.save:将数组以二进制格式保存到磁盘，扩展名为.npy 。

√.item()：遍历（键值对）。

√tf.shape(a)和 a.get_shape()比较

相同点：都可以得到 tensor a 的尺寸

不同点：tf.shape()中 a 的数据类型可以是 tensor, list, array；而 a.get_shape()中 a 的数据类型只能是 tensor,且返回的是一个元组（tuple）。

例：
```
import tensorflow as tf
import numpy as np

x=tf.constant([[1,2,3],[4,5,6]]
y=[[1,2,3],[4,5,6]]
z=np.arange(24).reshape([2,3,4]))

sess=tf.Session()
# tf.shape()
```

```
    x_shape=tf.shape(x)          #  x_shape 是一个 tensor
    y_shape=tf.shape(y)          #  <tf.Tensor 'Shape_2:0'  shape=(2,)
dtype=int32>
    z_shape=tf.shape(z)          #  <tf.Tensor 'Shape_5:0' shape=(3,)
dtype=int32>
    print sess.run(x_shape)      # 结果:[2 3]
    print sess.run(y_shape)      # 结果:[2 3]
    print sess.run(z_shape)      # 结果:[2 3 4]


    #a.get_shape()
    x_shape=x.get_shape()        # 返回的是 TensorShape([Dimension(2),
Dimension(3)]),不能使用 sess.run(), 因为返回的不是 tensor 或 string,而
是元组
    x_shape=x.get_shape().as_list()  # 可以使用 as_list()得到具体的尺
寸, x_shape=[2 3]
    y_shape=y.get_shape()          # AttributeError: 'list' object has
no attribute 'get_shape'
    z_shape=z.get_shape()          # AttributeError: 'numpy.ndarray'
object has no attribute 'get_shape'
```

√tf.nn.bias_add(乘加和，bias)：把 bias 加到乘加和上。

√tf.reshape(tensor, shape)：

改变 tensor 的形状

```
# tensor ‘t’ is [1, 2, 3, 4, 5, 6, 7, 8, 9]
# tensor ‘t’ has shape [9]
reshape(t, [3, 3]) ==>
[[1, 2, 3],
[4, 5, 6],
[7, 8, 9]]
```

#如果 shape 有元素[-1],表示在该维度打平至一维

-1 将自动推导得为 9:

reshape(t, [2, -1]) ==>

[[1, 1, 1, 2, 2, 2, 3, 3, 3],

[4, 4, 4, 5, 5, 5, 6, 6, 6]]

√np.argsort(列表)：对列表从小到大排序。

√OS 模块

os.getcwd()：返回当前工作目录。

os.path.join(path1[,path2[,.....]])：

返回值：将多个路径组合后返回。

注意：第一个绝对路径之前的参数将被忽略。

例：

```
>>>  import  os
>>> vgg16_path = os.path.join(os.getcwd(),"vgg16.npy")
#当前目录/vgg16.npy，索引到 vgg16.npy 文件
```

√np.save：写数组到文件（未压缩二进制形式），文件默认的扩展名是.npy 。

np.save("名.npy"，某数组):将某数组写入“名.npy”文件。

某变量 = np.load("名.npy"，encoding = " ").item()：将“名.npy”文件读出给某变量。encoding = " " 可以不写‘latin1’、‘ASCII’、‘bytes’，默认为’ASCII’。

例：

```
>>> import numpy as np
A = np.arange(15).reshape(3,5)
>>> A
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> np.save("A.npy",A)    #如果文件路径末尾没有扩展名.npy，该扩展名会被自动加上。
>>> B=np.load("A.npy")
```

```
>>> B
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

√tf.split(dimension, num_split, input)：

dimension：输入张量的哪一个维度，如果是 0 就表示对第 0 维度进行切割。

num_split：切割的数量，如果是 2 就表示输入张量被切成 2 份，每一份是一个列表。

例：

```
import tensorflow as tf;
import numpy as np;

A = [[1,2,3],[4,5,6]]
x = tf.split(1, 3, A)

with tf.Session() as sess:
    c = sess.run(x)
    for ele in c:
        print ele
```

输出：

```
[[1]
  [4]]
[[2]
  [5]]
[[3]
  [6]]
```

√tf.concat(concat_dim, values)：

沿着某一维度连结 tensor：

```
t1 = [[1, 2, 3], [4, 5, 6]]
```

t2 = [[7, 8, 9], [10, 11, 12]]

tf.concat(0, [t1, t2]) ==> [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]

tf.concat(1, [t1, t2]) ==> [[1, 2, 3, 7, 8, 9], [4, 5, 6, 10, 11, 12]]

如果想沿着 tensor 一新轴连结打包,那么可以：

tf.concat(axis, [tf.expand_dims(t, axis) for t in tensors])

等同于 tf.pack(tensors, axis=axis)

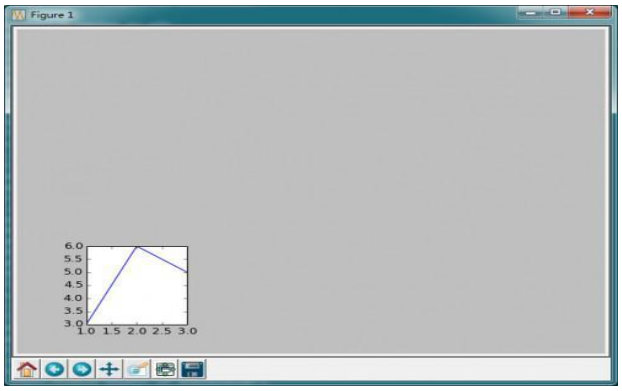
√fig = plt.figure("图名字")：实例化图对象。

√ax = fig.add_subplot(m n k)：将画布分割成 m 行 n 列，图像画在从左到右从上到下的第 k 块。

例：

```
#引入对应的库函数
import matplotlib.pyplot as plt
from numpy import *

#绘图
fig = plt.figure()
ax = fig.add_subplot(3 4 9)
ax.plot(x,y)
plt.show()
```



ax.bar(bar 的个数，bar 的值，每个 bar 的名字，bar 的宽，bar 的颜色)：绘制直方图。给出 bar 的个数，bar 的值，每个 bar 的名字，bar 的宽，bar 的颜色。

ax.set_ylabel("")：给出 y 轴的名字。

ax.set_title("")：给出子图的名字。

ax.text(x,y,string,fontsize=15,verticalalignment="top",horizontalali
gnment="right")：

x,y：表示坐标轴上的值。

string：表示说明文字。

fontsize：表示字体大小。

verticalalignment：垂直对齐方式，参数：[‘center’ | ‘top’ | ‘bottom’
| ‘baseline’]

horizontalalignment：水平对齐方式，参数：[‘center’ | ‘right’ | ‘left’]

xycoords 选择指定的坐标轴系统:

- figure points

  points from the lower left of the figure 点在图左下方
- figure pixels

  pixels from the lower left of the figure 图左下角的像素
- figure fraction

  fraction of figure from lower left 左下角数字部分
- axes points

  points from lower left corner of axes 从左下角点的坐标
- axes pixels

  pixels from lower left corner of axes 从左下角的像素坐标
- axes fraction

  fraction of axes from lower left 左下角部分
- data

  use the coordinate system of the object being annotated(default) 使用的坐标系统被注释的对象（默认）
- polar(theta,r)
- if not native ‘data’ coordinates t arrowprops #箭头参数,参数类型为字典 dict

- width

  the width of the arrow in points 点箭头的宽度
- headwidth

  the width of the base of the arrow head in points 在点的箭头底座的宽度
- headlength

  the length of the arrow head in points 点箭头的长度
- shrink

  fraction of total length to ‘shrink’ from both ends 总长度为分数“缩水”从两端
- facecolor 箭头颜色

bbox 给标题增加外框 ， 常用参数如下：

- boxstyle 方框外形
- facecolor(简写 fc)背景颜色
- edgecolor(简写 ec)边框线条颜色
- edgewidth 边框线条大小

bbox=dict(boxstyle='round,pad=0.5',fc='yellow',ec='k',lw=1 ,alpha=0.5) #fc 为 facecolor,ec 为 edgecolor,lw 为 lineweight

√plt.show()：画出来。

√axo = imshow(图)：画子图。

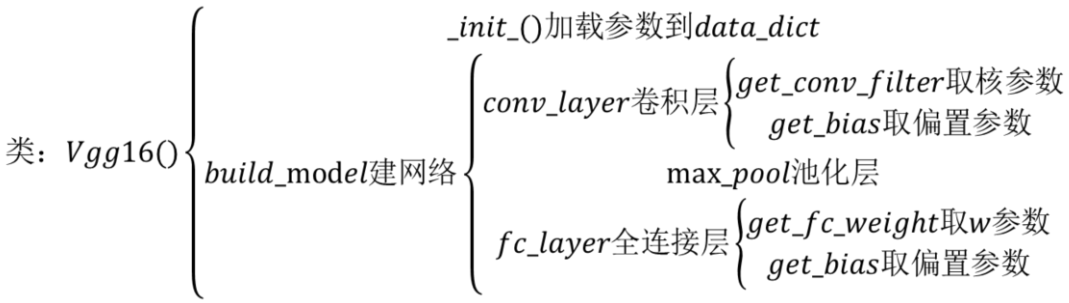
图 = io.imread(图路径索引到文件)。

√vgg 网络具体结构

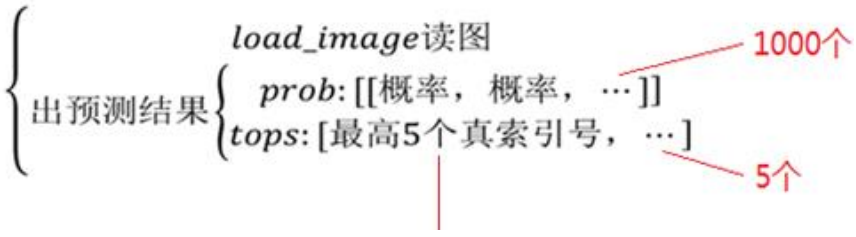
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

http://blog.csdn.net/qm

√ vgg16.py 还原网络和参数



√ app.py 读入待判图，给出可视化结果



代码中 i 为其真索引号，对应 top5[0]

三、课程中 VGG 源码的全文注释

vgg16.py

```
#!/usr/bin/python

#coding:utf-8


import inspect
```

```
import os
import numpy as np
import tensorflow as tf
import time
import matplotlib.pyplot as plt

VGG_MEAN = [103.939, 116.779, 123.68] # 样本 RGB 的平均值

class Vgg16():
    def __init__(self, vgg16_path=None):
        if vgg16_path is None:
            vgg16_path = os.path.join(os.getcwd(), "vgg16.npy") # os.getcwd() 方法用于返回当前工作目录。
            print(vgg16_path)
            self.data_dict = np.load(vgg16_path, encoding='latin1').item() # 遍历其内键值对，导入模型参数

        for x in self.data_dict:  #遍历 data_dict 中的每个键
            print x

    def forward(self, images):

        # plt.figure("process pictures")
        print("build model started")
        start_time = time.time() # 获取前向传播的开始时间
        rgb_scaled = images * 255.0 # 逐像素乘以 255.0（根据原论文所述的初始化步骤）
        # 从 GRB 转换色彩通道到 BGR，也可使用 cv 中的 GRBtoBGR
        red, green, blue = tf.split(rgb_scaled,3,3)
        assert red.get_shape().as_list()[1:] == [224, 224, 1]
        assert green.get_shape().as_list()[1:] == [224, 224, 1]
        assert blue.get_shape().as_list()[1:] == [224, 224, 1]
        # 以上 assert 都是加入断言，用来判断每个操作后的维度变化是否和预期一致
```

```
bgr = tf.concat([
    blue - VGG_MEAN[0],
    green - VGG_MEAN[1],
    red - VGG_MEAN[2]],3)
# 逐样本减去每个通道的像素平均值，这种操作可以移除图像的平均亮度值，该方法常用在灰度图像上
assert bgr.get_shape().as_list()[1:] == [224, 224, 3]

# 接下来构建 VGG 的 16 层网络（包含 5 段卷积，3 层全连接），并逐层根据命名空间读取网络参数
# 第一段卷积，含有两个卷积层，后面接最大池化层，用来缩小图片尺寸
self.conv1_1 = self.conv_layer(bgr, "conv1_1")
# 传入命名空间的 name，来获取该层的卷积核和偏置，并做卷积运算，最后返回经过经过激活函数后的值
self.conv1_2 = self.conv_layer(self.conv1_1, "conv1_2")
# 根据传入的 pooling 名字对该层做相应的池化操作
self.pool1 = self.max_pool_2x2(self.conv1_2, "pool1")

# 下面的前向传播过程与第一段同理
# 第二段卷积，同样包含两个卷积层，一个最大池化层
self.conv2_1 = self.conv_layer(self.pool1, "conv2_1")
self.conv2_2 = self.conv_layer(self.conv2_1, "conv2_2")
self.pool2 = self.max_pool_2x2(self.conv2_2, "pool2")

# 第三段卷积，包含三个卷积层，一个最大池化层
self.conv3_1 = self.conv_layer(self.pool2, "conv3_1")
self.conv3_2 = self.conv_layer(self.conv3_1, "conv3_2")
self.conv3_3 = self.conv_layer(self.conv3_2, "conv3_3")
self.pool3 = self.max_pool_2x2(self.conv3_3, "pool3")

# 第四段卷积，包含三个卷积层，一个最大池化层
self.conv4_1 = self.conv_layer(self.pool3, "conv4_1")
self.conv4_2 = self.conv_layer(self.conv4_1, "conv4_2")
```

```
        self.conv4_3 = self.conv_layer(self.conv4_2, "conv4_3")
        self.pool4 = self.max_pool_2x2(self.conv4_3, "pool4")

        # 第五段卷积，包含三个卷积层，一个最大池化层
        self.conv5_1 = self.conv_layer(self.pool4, "conv5_1")
        self.conv5_2 = self.conv_layer(self.conv5_1, "conv5_2")
        self.conv5_3 = self.conv_layer(self.conv5_2, "conv5_3")
        self.pool5 = self.max_pool_2x2(self.conv5_3, "pool5")

        # 第六层全连接
        self.fc6 = self.fc_layer(self.pool5, "fc6") # 根据命名空间 name 做加权求和运算
        assert self.fc6.get_shape().as_list()[1:] == [4096] # 4096 是该层输出后的长度
        self.relu6 = tf.nn.relu(self.fc6) # 经过 relu 激活函数

        # 第七层全连接，和上一层同理
        self.fc7 = self.fc_layer(self.relu6, "fc7")
        self.relu7 = tf.nn.relu(self.fc7)

        # 第八层全连接
        self.fc8 = self.fc_layer(self.relu7, "fc8")
        # 经过最后一层的全连接后，再做 softmax 分类，得到属于各类别的概率
        self.prob = tf.nn.softmax(self.fc8, name="prob")

        end_time = time.time() # 得到前向传播的结束时间
        print(("time consuming: %f" % (end_time-start_time)))

        self.data_dict = None # 清空本次读取到的模型参数字典

    # 定义卷积运算
    def conv_layer(self, x, name):
```

```
    with tf.variable_scope(name): # 根据命名空间找到对应卷积层的网络参数
        w = self.get_conv_filter(name) # 读到该层的卷积核
        conv = tf.nn.conv2d(x, w, [1, 1, 1, 1], padding='SAME') # 卷积计算
        conv_biases = self.get_bias(name) # 读到偏置项
        result = tf.nn.relu(tf.nn.bias_add(conv, conv_biases)) # 加上偏置，并做激活计算
        return result
# 定义获取卷积核的函数
def get_conv_filter(self, name):
    # 根据命名空间 name 从参数字典中取到对应的卷积核
    return tf.constant(self.data_dict[name][0], name="filter")
# 定义获取偏置项的函数
def get_bias(self, name):
    # 根据命名空间 name 从参数字典中取到对应的卷积核
    return tf.constant(self.data_dict[name][1], name="biases")
# 定义最大池化操作
def max_pool_2x2(self, x, name):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME', name=name)
# 定义全连接层的前向传播计算
def fc_layer(self, x, name):
    with tf.variable_scope(name): # 根据命名空间 name 做全连接层的计算
        shape = x.get_shape().as_list()  # 获取该层的维度信息列表
    print "fc_layer shape ",shape
        dim = 1
        for i in shape[1:]:
            dim *= i     # 将每层的维度相乘
         # 改变特征图的形状，也就是将得到的多维特征做拉伸操作，只在进入第六层全连接层做该操作
        x = tf.reshape(x, [-1, dim])

        w = self.get_fc_weight(name)# 读到权重值
        b = self.get_bias(name) # 读到偏置项值
```

```
            result = tf.nn.bias_add(tf.matmul(x, w), b) # 对该层输入做加权求和，再加上偏置
            return result
    # 定义获取权重的函数
    def get_fc_weight(self, name):  # 根据命名空间 name 从参数字典中取到对应的权重
        return tf.constant(self.data_dict[name][0], name="weights")
```

utils.py

```
#!/usr/bin/python
#coding:utf-8

from skimage import io, transform
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from pylab import mpl

mpl.rcParams['font.sans-serif']=['SimHei'] # 正常显示中文标签
mpl.rcParams['axes.unicode_minus']=False # 正常显示正负号

def load_image(path):
    fig = plt.figure("Centre and Resize")

    img = io.imread(path) # 根据传入的路径读入图片
    img = img / 255.0 # 将像素归一化到[0,1]

    # 将该画布分为一行三列
    ax0 = fig.add_subplot(131)   # 把下面的图像放在该画布的第一个位置
    ax0.set_xlabel(u'Original Picture') # 添加子标签
```

```
    ax0.imshow(img) #  添加展示该图像


    short_edge = min(img.shape[:2])    #  找到该图像的最短边
    y = (img.shape[0] - short_edge) / 2
    x = (img.shape[1] - short_edge) / 2 #  把图像的 w 和 h 分别减去最短边，并求平均
    crop_img = img[y:y+short_edge, x:x+short_edge] #  取出切分出的中心图像
    print crop_img.shape


    ax1 = fig.add_subplot(132) #  把下面的图像放在该画布的第二个位置
    ax1.set_xlabel(u"Centre Picture") #  添加子标签
    ax1.imshow(crop_img)


    re_img = transform.resize(crop_img, (224, 224)) # resize 成固定的 imag_szie


    ax2 = fig.add_subplot(133) #  把下面的图像放在该画布的第三个位置
    ax2.set_xlabel(u"Resize Picture") #  添加子标签
    ax2.imshow(re_img)
    img_ready = re_img.reshape((1, 224, 224, 3))
    return img_ready
#  定义百分比转换函数
def percent(value):
    return '%.2f%%' % (value * 100)
```

app.py

```
#coding:utf-8
import numpy as np
# Linux 服务器没有 GUI 的情况下使用 matplotlib 绘图，必须置于 pyplot 之前
#import matplotlib
#matplotlib.use('Agg')
```

```
import tensorflow as tf
import matplotlib.pyplot as plt

#  下面三个是引用自定义模块
import vgg16
import utils
from Nclasses import labels

img_path = raw_input('Input the path and image name:')
img_ready = utils.load_image(img_path) #  调用 load_image()函数，对待测试的图像做一些预处理操作

#定义一个 figure 画图窗口，并指定窗口的名称，也可以设置窗口修的大小
fig=plt.figure(u"Top-5  预测结果")

with tf.Session() as sess:
    #  定义一个维度为[1,224,224,3],类型为 float32 的 tensor 占位符
    x = tf.placeholder(tf.float32, [1, 224, 224, 3])
    vgg = vgg16.Vgg16() #  类 Vgg16 实例化出 vgg
    #  调用类的成员方法 forward()，并传入待测试图像，这也就是网络前向传播的过程
    vgg.forward(x)
    #  将一个 batch 的数据喂入网络，得到网络的预测输出
    probablity = sess.run(vgg.prob, feed_dict={x:img_ready})

    # np.argsort 函数返回预测值（probability 的数据结构[[各预测类别的概率值]]）由小到大的索引值，
    #  并取出预测概率最大的五个索引值
    top5 = np.argsort(probability[0])[-1:-6:-1]
    print "top5:",top5
```

```
    # 定义两个 list---对应的概率值和实际标签（zebra）
    values = []
    bar_label = []

    for n, i in enumerate(top5): # 枚举上面取出的五个索引值
        print "n:",n
        print "i:",i
        values.append(probability[0][i]) # 将索引值对应的预测概率值取出并放入 values
        bar_label.append(labels[i]) # 根据索引值取出对应的实际标签并放入 bar_label
        print i, ":", labels[i], "----", utils.percent(probability[0][i]) # 打印属于某个类别的概率

    ax = fig.add_subplot(111) # 将画布划分为一行一列，并把下图放入其中
    # bar()函数绘制柱状图，参数 range(len(values)是柱子下标，values 表示柱高的列表（也就是五个预测概率值，
    # tick_label 是每个柱子上显示的标签（实际对应的标签），width 是柱子的宽度，fc 是柱子的颜色）
    ax.bar(range(len(values)), values, tick_label=bar_label, width=0.5, fc='g')
    ax.set_ylabel(u'probability') # 设置横轴标签
    ax.set_title(u'Top-5') # 添加标题
    for a,b in zip(range(len(values)), values):
        # 在每个柱子的顶端添加对应的预测概率值，a，b 表示坐标，b+0.0005 表示要把文本信息放置在高于每个柱子顶端
0.0005 的位置，
        # center 是表示文本位于柱子顶端水平方向上的的中间位置，bottom 是将文本水平放置在柱子顶端垂直方向上的底端
位置，fontsize 是字号
        ax.text(a, b+0.0005, utils.percent(b), ha='center', va = 'bottom', fontsize=7)
    plt.savefig('./result.jpg') # 保存图片
    plt.show() # 弹窗展示图像（linux 服务器上将该句注释掉）
```

四、课程中提到的练习内容

打印出 img_ready 的维度：

app.py 第 11 行加入 print "img_ready shape", tf.Session().run(tf.shape(img_ready))

```
#coding:utf-8
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import vgg16
import utils
from Nclasses import labels

img_path = raw_input('Input the path and image name:')
img_ready = utils.load_image(img_path) #[1, 224, 224, 3)]
print "img_ready shape", tf.Session().run(tf.shape(img_ready))

```

输出：img_ready shape [1 224 224 3]

上下文管理器

```
with tf.variable_scope("foo"):
    with tf.variable_scope("bar"):
        v = tf.get_variable("v", [1])
        assert v.name == "foo/bar/v:0"
```

出现命名层次结构 foo/bar/v