读书笔记 5 逐层可视化图像特征

2014.7.23 薛开宇

本篇比较重要，讲述了如何可视化每层的特征。其中有些涉及分类的细节，该部分具体看读书笔记 6。

注意，本笔记的代码由原文复制过来，再复制过去 ipython 可能会有些格式上的错误，建议使用原文（本文最后）的代码进行复制。

1.1 开始 Ipython notebook

配置 python

文件夹 python 的 requirements.txt 列举了 Caffe 依赖的 python 库，可以看一下教程具体配置网上有教程 http://www.cnblogs.com/alfredtofu/p/3577241.html

我也是手动安装的

Cython>=0.19.2
h5py>=2.2.0
ipython>=1.1.0
leveldb>=0.191
matplotlib>=1.3.1
networkx>=1.8.1
nose>=1.3.0
numpy>=1.7.1
pandas>=0.12.0
protobuf>=2.5.0
python-gflags>=2.0
scikit-image>=0.9.3
scikit-learn>=0.14.1
scipy>=0.13.2

以 Cython 为例，手动的话用 sudo pip install Cython 即可，然后在 pip 的过程中有什么问题就相应解决。

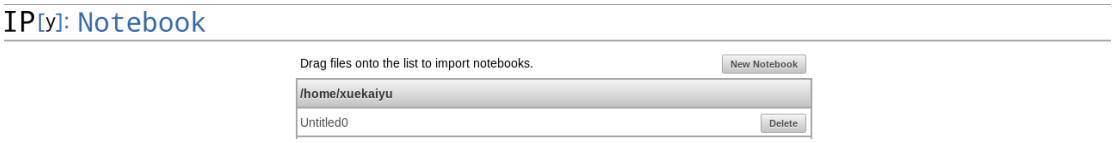
同时注意的是，我们下面的语句都是用 ipython，ipython 中用 ！表示执行 shell 命令，用 $ 将 python 的变量转化成 shell 变量。通过这种两个符号，我们就可以做到和 shell 命令之间的交互，可以非常方便地做许多复杂的工作。

Ipython 可以在终端执行，但为了方便，我们可以用 Ipython notebook。notebook 是 web based IPython 封装，但是可以展现富文本，使得整个工作可以以笔记的形式展现、存储，对于交互编程、学习非常方便。Ipython notebook 的安装配置可以在网上找到教程。

http://blog.csdn.net/awayyao/article/details/17785321

配置好 python，ipython 和 python notebook，直接在终端输入：

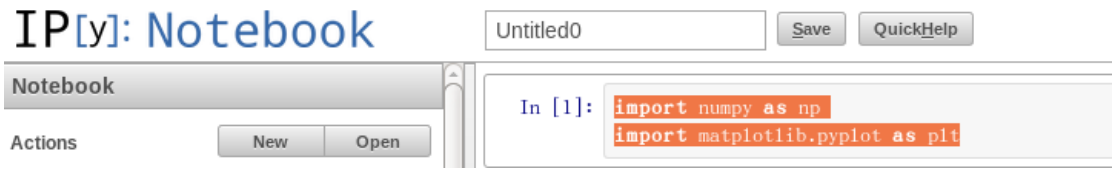
ipython notebook



1.2 开始进行

先输入一段代码：

```
In [1]:  import numpy as np
         import matplotlib.pyplot as plt
```



按 shift+enter 执行。

代码的意思是调入 numpy 子程序，调入后名字为 np，调入 matplotlib.pyplot 子程序（用作画图），调入后名字为 plt。

然后是 （以下是我的路径）：

```
caffe_root = '/home/xuekaiyu/caffe-master/'
```

这里注意路径一定要设置正确，记得前后可能都有“/”，路径的使用是{caffe_root}/examples，记得 caffe-master 中的 python 文件夹需要包括 caffe 文件夹。

接着是

```
import sys
sys.path.insert(0, caffe_root + 'python')
import caffe
```

把 ipython 的路径改到我们之前指定的地方，以便可以调入 caffe 模块，如果不改路径，import 这个指令只会在当前目录查找，是找不到 caffe 的。

接着是：

```
plt.rcParams['figure.figsize'] = (10, 10)
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
```

代码的意思是，显示的图表大小为 10，图形的插值是以最近为原则，图像颜色是灰色

```
net = caffe.Classifier(caffe_root + 'examples/imagenet/imagenet_deploy.prototxt',
        caffe_root + 'examples/imagenet/caffe_reference_imagenet_model')
```

预先下载好模型，模型的大小 232.57MB,可以用指令：examples/imagenet/get_caffe_reference_imagenet_model.sh. 下载。调用分类网络进行分类，记得设置好路径，启用后，终端将用该模型进行分类。

```
I0722 15:41:23.833319  6558 net.cpp:99] relu7 -> fc7 (in-place)
I0722 15:41:23.833329  6558 net.cpp:126] Top shape: 10 4096 1 1 (40960)
I0722 15:41:23.833338  6558 net.cpp:152] relu7 needs backward computation.
I0722 15:41:23.833364  6558 net.cpp:75] Creating Layer drop7
I0722 15:41:23.833375  6558 net.cpp:85] drop7 <- fc7
I0722 15:41:23.833386  6558 net.cpp:99] drop7 -> fc7 (in-place)
I0722 15:41:23.833398  6558 net.cpp:126] Top shape: 10 4096 1 1 (40960)
I0722 15:41:23.833410  6558 net.cpp:152] drop7 needs backward computation.
I0722 15:41:23.833423  6558 net.cpp:75] Creating Layer fc8
I0722 15:41:23.833433  6558 net.cpp:85] fc8 <- fc7
I0722 15:41:23.833444  6558 net.cpp:111] fc8 -> fc8
I0722 15:41:23.841811  6558 net.cpp:126] Top shape: 10 1000 1 1 (10000)
I0722 15:41:23.841853  6558 net.cpp:152] fc8 needs backward computation.
I0722 15:41:23.841869  6558 net.cpp:75] Creating Layer prob
I0722 15:41:23.841897  6558 net.cpp:85] prob <- fc8
I0722 15:41:23.841910  6558 net.cpp:111] prob -> prob
I0722 15:41:23.841934  6558 net.cpp:126] Top shape: 10 1000 1 1 (10000)
I0722 15:41:23.841945  6558 net.cpp:152] prob needs backward computation.
I0722 15:41:23.841958  6558 net.cpp:163] This network produces output prob
I0722 15:41:23.841984  6558 net.cpp:181] Collecting Learning Rate and Weight Dec
ay.
I0722 15:41:23.842002  6558 net.cpp:174] Network initialization done.
I0722 15:41:23.842013  6558 net.cpp:175] Memory required for Data 42022840
```

接着：输入以下指令：

```
net.set_phase_test()
net.set_mode_cpu()
```

再接着，是以下指令：

```
net.set_mean('data', caffe_root + 'python/caffe/imagenet/ilsvrc_2012_mean.npy')  # ImageNet 的均值
net.set_channel_swap('data', (2,1,0))  # 因为参考模型本来频道是 BGR，所以要将 RGB 转换
net.set_input_scale('data', 255)  #参考模型运行在【0，255】的灰度，而不是【0，1】。
```

接着是：

```
scores = net.predict([caffe.io.load_image(caffe_root + 'examples/images/cat.jpg')])
```

出现 in classifier.py line,TpyeError: squeeze takes no keyword arguments. 纠结了半天之后，发现是 numpy 版本过低导致的，ubuntu 自带的 numpy 版本是 1.6.1，更新到 1.7 以上。

用指令：sudo pip install numpy --upgrade

然后输入：

```
[(k, v.data.shape) for k, v in net.blobs.items()]
```

显示出各层的参数和形状，第一个是批次，第二个 feature map 数目，第三和第四是每个神经元中图片的长和宽，可以看出，输入是 227*227 的图片，三个频道，卷积是 32 个卷积核卷三个频道，因此有 96 个 feature map

Out[4]:

```
[('data', (10, 3, 227, 227)),
 ('conv1', (10, 96, 55, 55)),
 ('pool1', (10, 96, 27, 27)),
 ('norm1', (10, 96, 27, 27)),
 ('conv2', (10, 256, 27, 27)),
 ('pool2', (10, 256, 13, 13)),
 ('norm2', (10, 256, 13, 13)),
 ('conv3', (10, 384, 13, 13)),
 ('conv4', (10, 384, 13, 13)),
 ('conv5', (10, 256, 13, 13)),
 ('pool5', (10, 256, 6, 6)),
 ('fc6', (10, 4096, 1, 1)),
 ('fc7', (10, 4096, 1, 1)),
 ('fc8', (10, 1000, 1, 1)),
 ('prob', (10, 1000, 1, 1))]
```

再输入：

```
[(k, v[0].data.shape) for k, v in net.params.items()]
predictions = out[self.outputs[0]].squeeze(axis=(2,3))
```

输出：一些网络的参数

```
[('conv1', (96, 3, 11, 11)),
 ('conv2', (256, 48, 5, 5)),
 ('conv3', (384, 256, 3, 3)),
 ('conv4', (384, 192, 3, 3)),
 ('conv5', (256, 192, 3, 3)),
 ('fc6', (1, 1, 4096, 9216)),
 ('fc7', (1, 1, 4096, 4096)),
 ('fc8', (1, 1, 1000, 4096))]
```

再接着是一些显示图片的指令：

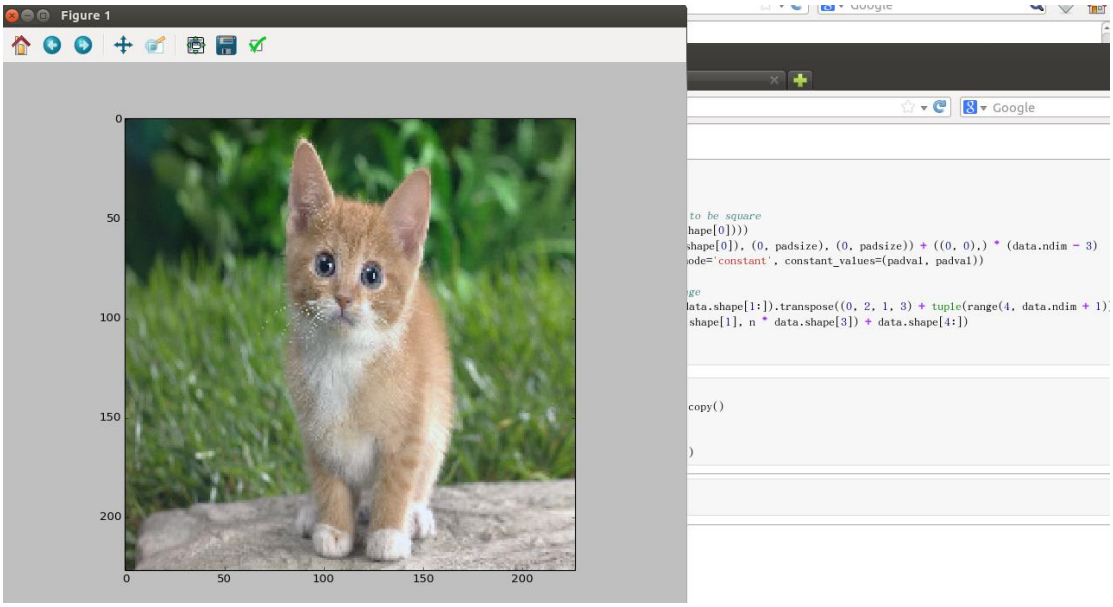
```
    # our network takes BGR images, so we need to switch color channels
    #  网络需要的是 BGR 格式图片，所以转换颜色频道
def showimage(im):
    if im.ndim == 3:
    m = im[:, :, ::-1]
  plt.imshow(im)

    # take an array of shape (n, height, width) or (n, height, width, channels)
    #  用一个格式是（数量，高，宽）或（数量，高，宽，频道）的阵列
    #    and visualize each (height, width) thing in a grid of size approx. sqrt(n) by sqrt(n)
    #  每个可视化的都是在一个由一个个网格组成
```

```
def vis_square(data, padsize=1, padval=0):
        data -= data.min()
        data /= data.max()
            # force the number of filters to be square
n = int(np.ceil(np.sqrt(data.shape[0])))
padding = ((0, n ** 2 - data.shape[0]), (0, padsize), (0, padsize)) + ((0, 0),) * (data.ndim - 3)
data = np.pad(data, padding, mode='constant', constant_values=(padval, padval))

            # 对图像使用滤波器
data = data.reshape((n, n) + data.shape[1:]).transpose((0, 2, 1, 3) + tuple(range(4, data.ndim + 1)))
data = data.reshape((n * data.shape[1], n * data.shape[3]) + data.shape[4:])

        showimage(data)
```

接着输入：

```
# index four is the center crop
# 输出输入的图像
image = net.blobs['data'].data[4].copy()
image -= image.min()
image /= image.max()
showimage(image.transpose(1, 2, 0))
```

使用 ipt.show（）观看图像：



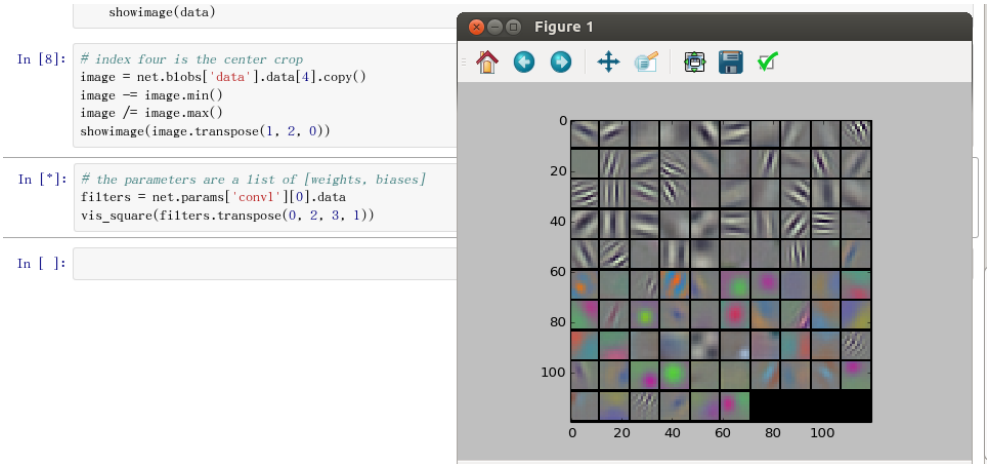
1.3 各层的特征：

第一个卷积层：

共 96 个过滤器

```
# the parameters are a list of [weights, biases]
filters = net.params['conv1'][0].data
vis_square(filters.transpose(0, 2, 3, 1))
```

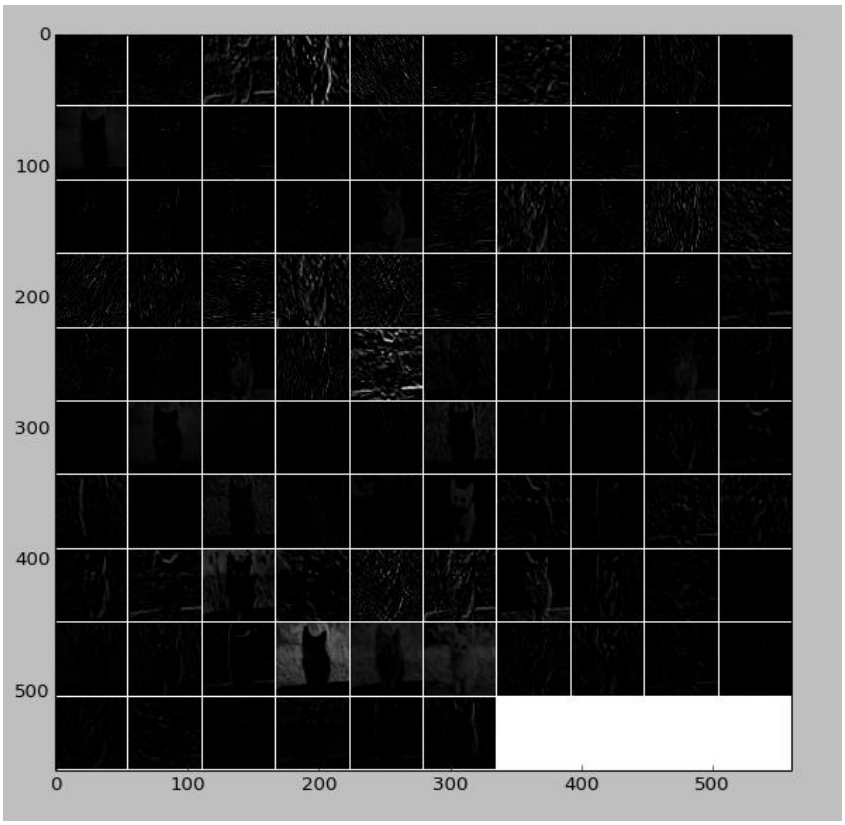
使用 ipt.show（）观看图像：



过滤后的输出，96 张 featuremap

```
feat = net.blobs['conv1'].data[4, :96]
vis_square(feat, padval=1)
```

使用 ipt.show（）观看图像：



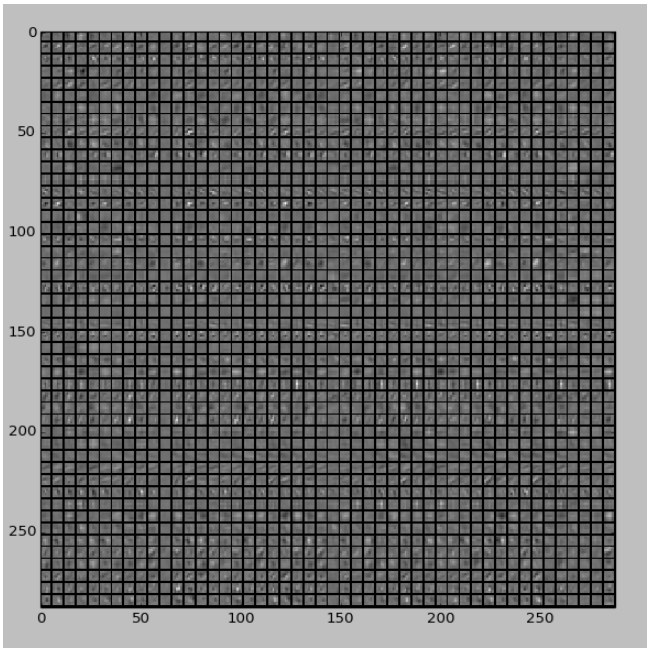
第二个卷积层：

有 128 个滤波器，每个尺寸为 5X5X48。我们只显示前面 48 个滤波器，每一个滤波器为一行。

输入：

```
filters = net.params['conv2'][0].data
vis_square(filters[:48].reshape(48**2, 5, 5))
```

使用 ipt.show（）观看图像：

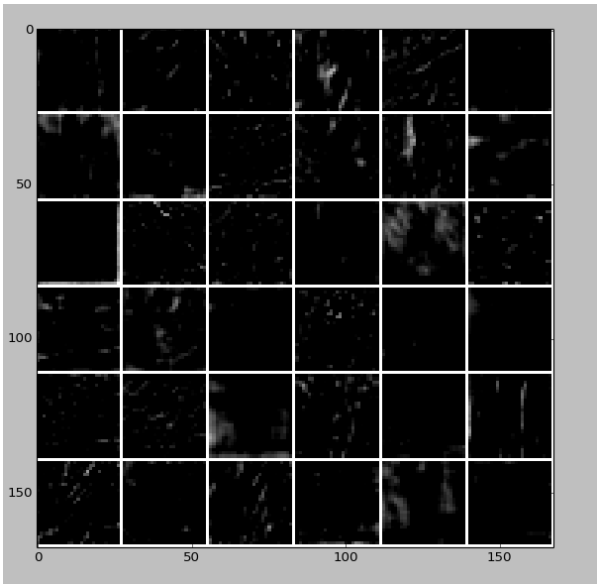


第二层输出 256 张 feature，这里显示 36 张。

输入：

```
feat = net.blobs['conv2'].data[4, :36]
vis_square(feat, padval=1)
```

使用 ipt.show（）观看图像：



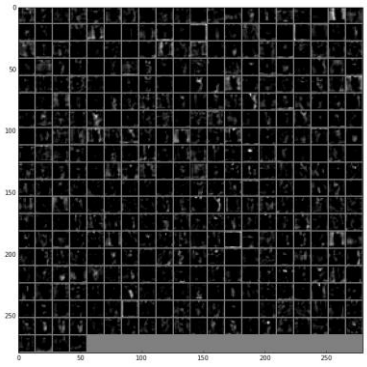
第三个卷积层：

全部 384 个 feature map

输入：

```
feat = net.blobs['conv3'].data[4]
vis_square(feat, padval=0.5)
```

使用 ipt.show（）观看图像：



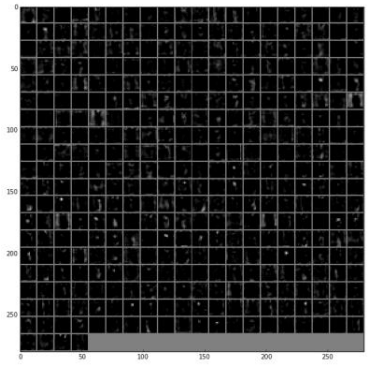
第四个卷积层：

全部 384 个 feature map

输入：

```
feat = net.blobs['conv4'].data[4]
vis_square(feat, padval=0.5)
```

使用 ipt.show（）观看图像：



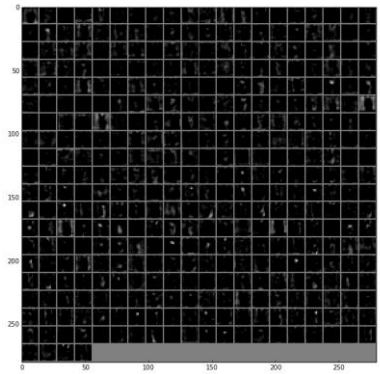
第五个卷积层：

全部 256 个 feature map

输入：

```
feat = net.blobs['conv5'].data[4]
vis_square(feat, padval=0.5)
```

使用 ipt.show（）观看图像：



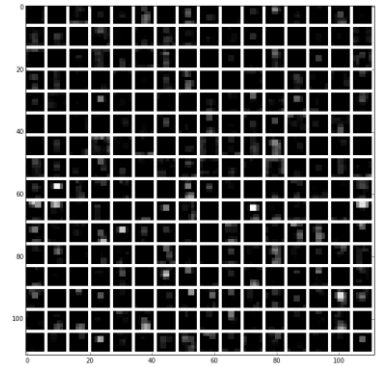
第五个 pooling 层：

我们也可以观察 pooling 层

输入：

```
feat = net.blobs['pool5'].data[4]
vis_square(feat, padval=1)
```

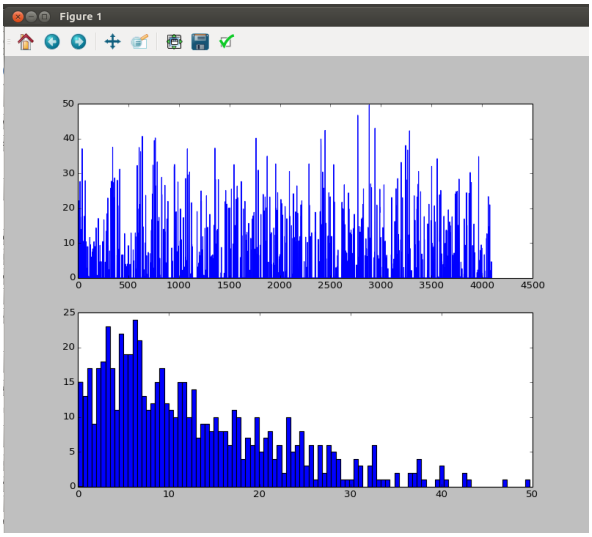
使用 ipt.show（）观看图像：



然后我们看看第六层输出后的直方分布：

```
feat = net.blobs['fc6'].data[4]
plt.subplot(2, 1, 1)
plt.plot(feat.flat)
plt.subplot(2, 1, 2)
_ = plt.hist(feat.flat[feat.flat > 0], bins=100)
```

使用 ipt.show（）观看图像：

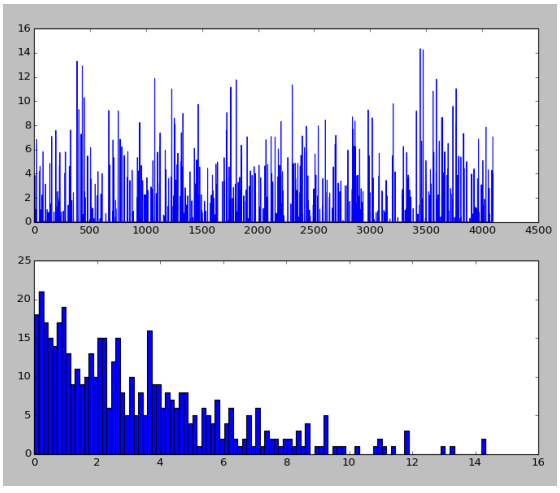


第七层输出后的直方分布：

可以看出值的分布没有这么平均了。

```
feat = net.blobs['fc7'].data[4]
plt.subplot(2, 1, 1)
plt.plot(feat.flat)
plt.subplot(2, 1, 2)
_ = plt.hist(feat.flat[feat.flat > 0], bins=100)
```

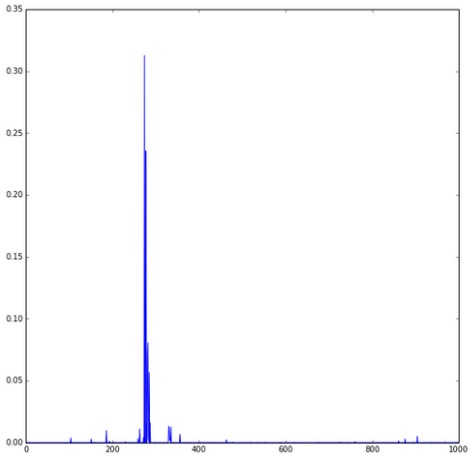
使用 ipt.show（）观看图像：



对后输出后的直方分布：

```
feat = net.blobs['prob'].data[4]
plt.subplot(2, 1, 1)
plt.plot(feat.flat)
```

使用 ipt.show（）观看图像：



最后看看标签：

```
imagenet_labels_filename = caffe_root + 'data/ilsvrc12/synset_words.txt'
try:
    labels = np.loadtxt(imagenet_labels_filename, str, delimiter='\t')
except:
    !../data/ilsvrc12/get_ilsvrc_aux.sh
    labels = np.loadtxt(imagenet_labels_filename, str, delimiter='\t')
top_k = net.blobs['prob'].data[4].flatten().argsort()[-1:-6:-1]
print labels[top_k]
```

```
In [41]: top_k = net.blobs['prob'].data[4].flatten().argsort()[-1:-6:-1]
         print labels[top_k]

['n02115913 dhole, Cuon alpinus' 'n02119022 red fox, Vulpes vulpes'
 'n02119789 kit fox, Vulpes macrotis' 'n02123159 tiger cat'
 'n02123045 tabby, tabby cat']
```

可以看出正确率还是挺高的。

资料来源：caffe 官网。

http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/filter_visualization.ipynb