读书笔记 7 如何改变模型参数将提取出的大量特征

用全卷积的分类器表示

2014.7.24 薛开宇

本篇主要是介绍一种方法，使输出不是整个图分类的概率，而是显示输入图各个部分的特征属于什么分类。

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）的代码进行复制使用。

1.1 前言

Caffe 可以通过编辑网络优化参数转化成我们所需要的模型。在这个例子，我们转化产品的内在分类器为卷积层。这会产生一个生成给定输入大小的分类图而不是单一的分类的全卷积模型。**本例子的一个分类会由每 6X6 区域的 pool5 层组成，用例子的 454x454 的输入尺寸产生一个 8X8 的分类图，如：**

```
array([[282, 282, 282, 282, 282, 278, 278, 278],
       [282, 283, 281, 281, 281, 281, 278, 282],
       [283, 283, 283, 283, 283, 287, 259, 278],
       [283, 283, 283, 283, 283, 259, 259, 259],
       [283, 283, 283, 283, 283, 283, 259, 259],
       [283, 283, 283, 283, 283, 283, 259, 259],
       [283, 283, 283, 283, 283, 259, 259, 852],
       [283, 283, 283, 283, 283, 263, 263, 331]])
```

其中，282 表示斑猫，281 表示波斯猫，有各种分类。

而原来的模型，只是给定某一个分类的概率。如：

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

这里是列出了概率排名前 5 的分类。

需要注意的是，这个模型不适合 sliding-window 测试因为它是整个图像的分类训练。如果想进行 Sliding-window 的训练和调整，可以通过在真值和 loss 的基础上定义一个滑动窗口，这样一个 loss 图就会由每一个位置组成并且像往常一样做。（这是计划中的想法，由读者完成中）

1.2 比较

```
!diff /home/xuekaiyu/caffe-master/examples/imagenet/imagenet_full_conv.prototxt
/home/xuekaiyu/caffe-master/examples/imagenet/imagenet_deploy.prototxt
```

截取一段输入

```
151,152c171,172                    #这里的意思是第几行有差别。
<     name: "fc6-conv"             #对比第一个和第二个文件发现名字改变了
<     type: CONVOLUTION            #把内积层变成卷积层
---
>     name: "fc6"
>     type: INNER_PRODUCT
154,155c174,179
<     top: "fc6-conv"
<     convolution_param {          #因为变成卷积层，多了卷积的参数设置。
---
>     top: "fc6"
>     blobs_lr: 1
>     blobs_lr: 2
>     weight_decay: 1
>     weight_decay: 0
>     inner_product_param {
157d180
<       kernel_size: 6             #因为 pool5 的输出是 36，因此全连接卷积层也是
                                   6X6
```

从上面可以看出，唯一需要改变的就是去改变全连接的分类器内积层变成卷积层，并且使用正确的滤波器大小 6X6（由于参考模型的分类以 Pool5 的 36 个元素作为输入的），同时步长为 1 保证密集。请注意，层重新命名了以避免当图层命名为 pretrained model 时 caffe 去载入旧的参数。

1.3 比较

```
import caffe
# Load the original network and extract the fully-connected layers' parameters.
#载入传统的网络来提取全连接层的参数
net=caffe.Net(caffe_root+'examples/imagenet/imagenet_deploy.prototxt',
caffe_root+'examples/imagenet/caffe_reference_imagenet_model')
params = ['fc6', 'fc7', 'fc8']
# fc_params = {name: (weights, biases)}
#fc_params 调用的格式写法{name: (weights, biases)}
fc_params = {pr: (net.params[pr][0].data, net.params[pr][1].data) for pr in params}
for fc in params:
    print '{} weights are {} dimensional and biases are {} dimensional'.format(fc,
fc_params[fc][0].shape, fc_params[fc][1].shape)
```

输出：
fc6 weights are (1, 1, 4096, 9216) dimensional and biases are (1, 1, 1, 4096) dimensional
fc7 weights are (1, 1, 4096, 4096) dimensional and biases are (1, 1, 1, 4096) dimensional
fc8 weights are (1, 1, 1000, 4096) dimensional and biases are (1, 1, 1, 1000) dimensional

考虑到内部产生的参数的形状。权值和偏值的第 0 和第 1 个面积为 1，第 2 个，第 3 个权值面积是输出的尺寸和当是最后的偏置面积是输出尺寸时是输入的尺寸。

```
# Load the fully-convolutional network to transplant the parameters.
#  载入全连接网络去移植参数
net_full_conv=caffe.Net(caffe_root+'examples/imagenet/imagenet_full_conv.prototxt',
caffe_root+'examples/imagenet/caffe_reference_imagenet_model')
params_full_conv = ['fc6-conv', 'fc7-conv', 'fc8-conv']
# conv_params = {name: (weights, biases)}
conv_params = {pr: (net_full_conv.params[pr][0].data, net_full_conv.params[pr][1].data) for pr in
params_full_conv}

for conv in params_full_conv:
    print '{} weights are {} dimensional and biases are {} dimensional'.format(conv,
conv_params[conv][0].shape, conv_params[conv][1].shape)
```

输出：
fc6-conv weights are (4096, 256, 6, 6) dimensional and biases are (1, 1, 1, 4096) dimensional
fc7-conv weights are (4096, 4096, 1, 1) dimensional and biases are (1, 1, 1, 4096) dimensional
fc8-conv weights are (1000, 4096, 1, 1) dimensional and biases are (1, 1, 1, 1000) dimensional

卷积层的权重由输出 X 输入 X 高度 X 宽度的尺寸决定，为了匹配大小，我们需要把内部产生的维度化成频道 X 高 X 宽度的滤波器矩阵。

偏值和内连接层相同，让我们先移植因为实在没有重塑的需要。

举个例子说，

本来第 6 个全连接层，层的连接是输入 1 个神经元输出 1 个神经元，每个神经元高 4096 维，宽 9216 维，权重为 1X1X4096X9216。现在做成卷积层，必须和前面的权重一样，因此层的连接是输入 4096 个神经元，输出 256 个神经元，每个神经元高 6 维，宽 6 维，这样权重也是 4096X256X6X6=1X1X4096X9216。

开始转换

```
for pr, pr_conv in zip(params, params_full_conv):
conv_params[pr_conv][1][...] = fc_params[pr][1]
```

输出层的频道数目需要主导内连接和卷积的权重，所以参数通过从内积层重塑输入参数维度矢量转换为频道 X 高 X 宽的形状

输入：

```
for pr, pr_conv in zip(params, params_full_conv):
    out, in_, h, w = conv_params[pr_conv][0].shape
    W = fc_params[pr][0].reshape((out, in_, h, w))
    conv_params[pr_conv][0][...] = W
```

现在，存储新的模型：

```
net_full_conv.save('imagenet/caffe_imagenet_full_conv')
```

这是二进制格式。

1.4 分类

我们运行一张传统的图片运行该模型，他会生成一个 8X8 的区域标记。

```
# load input and configure preprocessing
im = caffe.io.load_image(caffe_root+'examples/images/cat.jpg')
plt.imshow(im)
net_full_conv.set_mean('data', caffe_root+'/python/caffe/imagenet/ilsvrc_2012_mean.npy')
net_full_conv.set_channel_swap('data', (2,1,0))
net_full_conv.set_input_scale('data', 255.0)
# make classification map by forward pass and show top prediction index per location
out = net_full_conv.forward_all(data=np.asarray([net_full_conv.preprocess('data', im)]))
out['prob'][0].argmax(axis=0)
```

```
array([[282, 282, 282, 282, 282, 278, 278, 278],
       [282, 283, 281, 281, 281, 281, 278, 282],
       [283, 283, 283, 283, 283, 287, 259, 278],
       [283, 283, 283, 283, 283, 259, 259, 259],
       [283, 283, 283, 283, 283, 283, 259, 259],
       [283, 283, 283, 283, 283, 283, 259, 259],
       [283, 283, 283, 283, 283, 259, 259, 852],
       [283, 283, 283, 283, 283, 263, 263, 331]])
```

。 分类图包括各种的猫，281 是波斯猫，282 是斑猫，283 是猫科动物，还有其它一些是狐狸还有其他动物。

用这种方式，全卷积层可以用在图像提取密集的特征，这比单看分类图自身更有用。

参考资料:

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