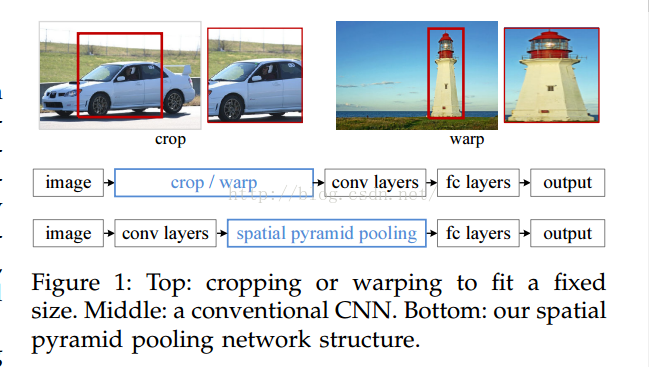
**空间金字塔池化**

**空间金字塔池化层简介：**

                  在对图片进行卷积操作的时候，卷积核的大小是不会发生变化的额，反向调节的权重仅仅是数值会发生变化。但是，但是，但是，输入的图片的大小你是否可以控制呢？哈哈，我们的输入图片大小是会发生变化的，这里图片大小的变化并不会在卷积操作和polling操作产生影响，但是会对全连接层的链接产生影响。这篇文章的核心就是解决如何对于不同的输入图片（主要是针对大小不同）都可以直接运用到已经训练好的网络中去。

**为什么要引入空间金字塔池化：**

          首先说一下为什么要有这个层：我们处理图片的大小不一，都有自己不同的像素值，但是同一批数据，如果非要进过一定的裁剪把他处理成为相同大小的图像，例如，我们可以先把图片的四个角裁剪下来，在加上一个中心区域的图片，这是五个变形的图片，然后再把图片水平翻转之后依然是相同的操作会得到五个图片，总合计是10个大小相同的图片，这是一种方法，当然还有其他的方法，例如在overFeat那篇论文中也提到一种方法，等等。这些裁剪技术都会达到不错的结果，但是依然会存在一些问题，例如，有些区域裁剪的时候都会有重复，无形之中加大了该区域的权重。所以，这篇paper就提出了金字塔池化来解决输入图片大小不一的情况。



     观察图一，最上面代表的是把原来的图片都进行适当的crop和wrap之后得到适当的像素值，中间的就是对应的网络模型。最下面的是这篇论文的模型，在最后一个卷积层后面紧随种恶一个spatital pyramid polling层，紧跟着的是全连接层。这样就可以解决不管输入图片的大小是多大，都可以用当前的网络进行测试的问题。

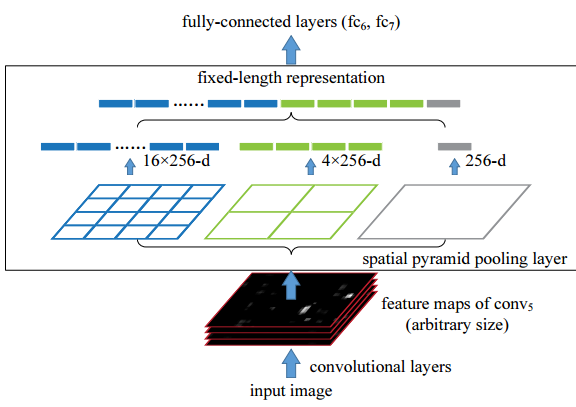
     但是我们需要明白的是为什么这个层放到了最后一个圈基层的后面呢？也就是说为什么convolution和polling都对如数图片的大小不敏感，而全连接层却敏感呢？我们来看看。假设输入图片的大小是100\*100，经过5个卷积核3\*3之后会差生5\*98\*98的feature maps，就算你的输入图片的大小变成102\*102，那我的feature maps就是5\*100\*100。这里的feature maps经过2\*2的polling之后得到的是25\*25和26\*26。没什么影响，这里的卷积核的大小是固定的，可以去卷积任何大小的图片。但是全连接层就不同了。假设最后一个卷积层有50个输出，下一层的全链接有1000个输入，那么这个链接矩阵就是50\*1000，哈哈，你想想，如果这里每次的输入图片大小都不一样，到这里如何进行链接呢？因为不同的图片大小经过最后一个卷积层的输入到输出之后压根就不可能都是50啊。这就是我们为什么要在全连接层这人进行操作的原因。

**空间金字塔池化层的特点**

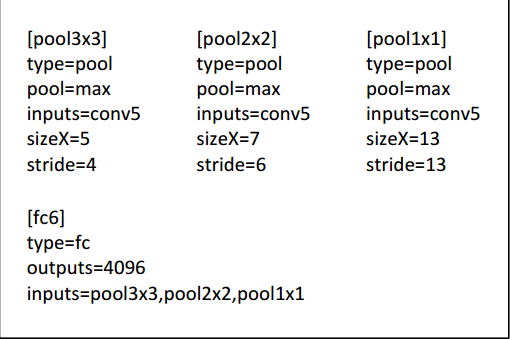
           当然，这篇paper的特点可不是仅仅有一个。金字塔池化层有如下的三个优点，第一：他可以解决输入图片大小不一造成的缺陷。第二：由于把一个feature map从不同的角度进行特征提取，再聚合的特点，显示了算法的robust的特性。第三：同时也在object recongtion增加了精度。其实，你也可以这样想，最牛掰的地方是因为在卷积层的后面对每一张图片都进行了多方面的特征提取，他就可以提高任务的精度。好比是不同大小的图片在不同的网络中进行训练一样，大大提高了模型的精度。SPP在现有的各种网络模型上都得到了state of the art 的高度，例如R-CNN上面。不仅仅如此，R-CNN需要对不同大小的边框内的图像feed into 不同的网络模型，整个过程是特别的耗时，SPP-Net刚好就可以决绝这个问题，大大的减少了时间。

**什么是金字塔池化层**

             哈哈，唧唧歪歪的说了半天，都没有说到重点上去，现在我们步入正题，说说什么是金字塔池化层。



           如上图所示，从下往上看，这是一个传统的网络架构模型，5层卷积层，这里的卷积层叫做convolution和pooling层的联合体，统一叫做卷积层，后面跟随全连接层。我们这里需要处理的就是在网络的全连接层前面加一层金字塔pooling层解决输入图片大小不一的情况。我们可以看到这里的spatital pyramid pooling layer就是把前一卷积层的feature maps的每一个图片上进行了3个卷积操作。最右边的就是原图像，中间的是把图像分成大小是4的特征图，最右边的就是把图像分成大小是16的特征图。那么每一个feature map就会变成16+4+1=21个feature maps。这不就解决了特征图大小不一的状况了吗？



       那么具体是如何操作的呢？我们来看看：图片的输入大小相等的时候，我们假设图片经过第五层的卷积核之后的输出大小是a\*a （例如，13\*13），我们bins的大小是n\*n,那么每一个窗口win=cell(a/n),然而stride的大小是stride=floor(a/n)，前面一个是向上取整，后面的是向下取整，最终会形成如上图所示的三个pooling操作。这三个的本质都是最大化池，只不过使用了不同的窗口大小和移动的步长而已。fc6代表的是全链接层。实验表明，多层次的卷积行为可以使得实验的最终精度提升。而不同大小的图片操作都是一样的方法。

**基于空间金字塔池化的卷积神经网络物体检测**

**原文地址：**<http://blog.csdn.net/hjimce/article/details/50187655>

**作者：**hjimce

**一、相关理论**

   本篇博文主要讲解大神何凯明2014年的paper：《Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition》，这篇paper主要的创新点在于提出了空间金字塔池化。paper主页：[http://research.microsoft.com/en-us/um/people/kahe/eccv14sppnet/index.html](http://research.microsoft.com/en-us/um/people/kahe/eccv14sppnet/index.html" \t "_blank)  这个**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)**比R-CNN算法的速度快了n多倍。

    我们知道在现有的CNN中，对于结构已经确定的网络，需要输入一张固定大小的图片，比如224\*224，32\*32,96\*96等。这样对于我们希望检测各种大小的图片的时候，需要经过裁剪，或者缩放等一系列操作，这样往往会降低识别检测的精度，于是paper提出了“空间金字塔池化”方法，这个算法的牛逼之处，在于使得我们构建的网络，可以输入任意大小的图片，不需要经过裁剪缩放等操作，只要你喜欢，任意大小的图片都可以。不仅如此，这个算法用了以后，精度也会有所提高，总之一句话：牛逼哄哄。

    空间金字塔池化，又称之为“SPP-Net”，记住这个名字，因为在以后的外文文献中，你会经常遇到，特别是物体检测方面的paper。这个就像什么：OverFeat、GoogleNet、R-CNN、AlexNet……为了方便，学完这篇paper之后，你就需要记住SPP-Net是什么东西了。空间金子塔以前在特征学习、特征表达的相关文献中，看到过几次这个算法。

   既然之前的CNN要求输入固定大小的图片，那么我们首先需要知道为什么CNN需要输入固定大小的图片？CNN大体包含3部分，卷积、池化、全连接。

首先是卷积，卷积操作对图片输入的大小会有要求吗？比如一个5\*5的卷积核，我输入的图片是30\*81的大小，可以得到(26,77)大小的图片，并不会影响卷积操作。我输入600\*500，它还是照样可以进行卷积，也就是卷积对图片输入大小没有要求，只要你喜欢，任意大小的图片进入，都可以进行卷积。

池化：池化对图片大小会有要求吗？比如我池化大小为（2，2）我输入一张30\*40的，那么经过池化后可以得到15\*20的图片。输入一张53\*22大小的图片，经过池化后，我可以得到26\*11大小的图片。因此池化这一步也没对图片大小有要求。只要你喜欢，输入任意大小的图片，都可以进行池化。

全连接层：既然池化和卷积都对输入图片大小没有要求，那么就只有全连接层对图片结果又要求了。因为全连接层我们的连接劝值矩阵的大小W，经过训练后，就是固定的大小了，比如我们从卷积到全连层，输入和输出的大小，分别是50、30个神经元，那么我们的权值矩阵（50,30）大小的矩阵了。因此空间金字塔池化，要解决的就是从卷积层到全连接层之间的一个过度。

也就是说在以后的文献中，一般空间金子塔池化层，都是放在卷积层到全连接层之间的一个网络层。

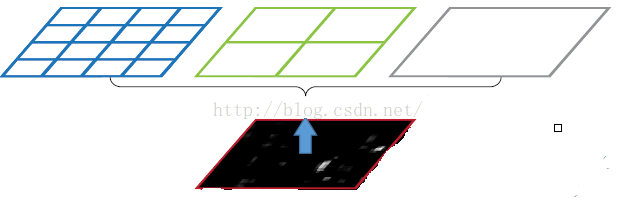
**二、算法概述**

OK,接着我们即将要讲解什么是空间金字塔池化。我们先从空间金字塔特征提取说起（这边先不考虑“池化”），空间金字塔是很久以前的一种特征提取方法，跟Sift、Hog等特征息息相关。为了简单起见，我们假设一个很简单两层网络：

输入层：一张任意大小的图片,假设其大小为(w,h)。

输出层：21个神经元。

也就是我们输入一张任意大小的特征图的时候，我们希望提取出21个特征。空间金字塔特征提取的过程如下：



图片尺度划分

如上图所示，当我们输入一张图片的时候，我们利用不同大小的刻度，对一张图片进行了划分。上面示意图中，利用了三种不同大小的刻度，对一张输入的图片进行了划分，最后总共可以得到16+4+1=21个块，我们即将从这21个块中，每个块提取出一个特征，这样刚好就是我们要提取的21维特征向量。

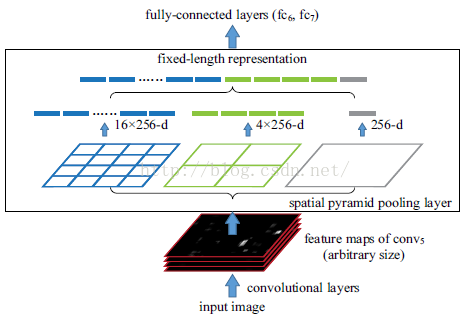
第一张图片,我们把一张完整的图片，分成了16个块，也就是每个块的大小就是(w/4,h/4);

第二张图片，划分了4个块，每个块的大小就是(w/2,h/2);

第三张图片，把一整张图片作为了一个块，也就是块的大小为(w,h)

空间金字塔最大池化的过程，其实就是从这21个图片块中，分别计算每个块的最大值，从而得到一个输出神经元。最后把一张任意大小的图片转换成了一个固定大小的21维特征（当然你可以设计其它维数的输出，增加金字塔的层数，或者改变划分网格的大小）。上面的三种不同刻度的划分，每一种刻度我们称之为：金字塔的一层，每一个图片块大小我们称之为：windows size了。如果你希望，金字塔的某一层输出n\*n个特征，那么你就要用windows size大小为：(w/n,h/n)进行池化了。

当我们有很多层网络的时候，当网络输入的是一张任意大小的图片，这个时候我们可以一直进行卷积、池化，直到网络的倒数几层的时候，也就是我们即将与全连接层连接的时候，就要使用金字塔池化，使得任意大小的特征图都能够转换成固定大小的特征向量，这就是空间金字塔池化的奥义（多尺度特征提取出固定大小的特征向量）。具体的流程图如下：



**三、算法源码实现**

 理论学的再多，终归要实践，实践是检验理论的唯一标准，caffe中有关于空间金字塔池化的源码，我这边就直接把它贴出来，以供学习使用，源码来自[https://github.com/BVLC/caffe](https://github.com/BVLC/caffe" \t "_blank)：

**[cpp]** [view plaincopy](http://blog.csdn.net/hjimce/article/details/50187655)

1. //1、输入参数pyramid\_level：表示金字塔的第几层。我们将对这一层，进行划分为2^n个图片块。金字塔从第0层开始算起，0层就是一整张图片
2. //第1层就是把图片划分为2\*2个块，第2层把图片划分为4\*4个块，以此类推……，也就是说我们块的大小就是[w/(2^n),h/(2^n)]
3. //2、参数bottom\_w、bottom\_h是我们要输入这一层网络的特征图的大小
4. //3、参数spp\_param是设置我们要进行池化的方法，比如最大池化、均值池化、概率池化……
5. LayerParameter SPPLayer<Dtype>::GetPoolingParam(**const** **int** pyramid\_level,
6. **const** **int** bottom\_h, **const** **int** bottom\_w, **const** SPPParameter spp\_param)
7. {
8. LayerParameter pooling\_param;
9. **int** num\_bins = pow(2, pyramid\_level);//计算可以划分多少个刻度，最后我们图片块的个数就是num\_bins\*num\_bins
10. //计算垂直方向上可以划分多少个刻度，不足的用pad补齐。然后我们最后每个图片块的大小就是(kernel\_w,kernel\_h)
11. **int** kernel\_h = ceil(bottom\_h / **static\_cast**<**double**>(num\_bins));//向上取整。采用pad补齐，pad的像素都是0
12. **int** remainder\_h = kernel\_h \* num\_bins - bottom\_h;
13. **int** pad\_h = (remainder\_h + 1) / 2;//上下两边分摊pad
14. //计算水平方向的刻度大小，不足的用pad补齐
15. **int** kernel\_w = ceil(bottom\_w / **static\_cast**<**double**>(num\_bins));
16. **int** remainder\_w = kernel\_w \* num\_bins - bottom\_w;
17. **int** pad\_w = (remainder\_w + 1) / 2;

20. pooling\_param.mutable\_pooling\_param()->set\_pad\_h(pad\_h);
21. pooling\_param.mutable\_pooling\_param()->set\_pad\_w(pad\_w);
22. pooling\_param.mutable\_pooling\_param()->set\_kernel\_h(kernel\_h);
23. pooling\_param.mutable\_pooling\_param()->set\_kernel\_w(kernel\_w);
24. pooling\_param.mutable\_pooling\_param()->set\_stride\_h(kernel\_h);
25. pooling\_param.mutable\_pooling\_param()->set\_stride\_w(kernel\_w);
27. **switch** (spp\_param.pool()) {
28. **case** SPPParameter\_PoolMethod\_MAX://窗口最大池化
29. pooling\_param.mutable\_pooling\_param()->set\_pool(
30. PoolingParameter\_PoolMethod\_MAX);
31. **break**;
32. **case** SPPParameter\_PoolMethod\_AVE://平均池化
33. pooling\_param.mutable\_pooling\_param()->set\_pool(
34. PoolingParameter\_PoolMethod\_AVE);
35. **break**;
36. **case** SPPParameter\_PoolMethod\_STOCHASTIC://随机概率池化
37. pooling\_param.mutable\_pooling\_param()->set\_pool(
38. PoolingParameter\_PoolMethod\_STOCHASTIC);
39. **break**;
40. **default**:
41. LOG(FATAL) << "Unknown pooling method.";
42. }
44. **return** pooling\_param;
45. }
47. **template** <**typename** Dtype>
48. //这个函数是为了获取我们本层网络的输入特征图、输出相关参数，然后设置相关变量，比如输入特征图的图片的大小、个数
49. **void** SPPLayer<Dtype>::LayerSetUp(**const** vector<Blob<Dtype>\*>& bottom,
50. **const** vector<Blob<Dtype>\*>& top) {
51. SPPParameter spp\_param = **this**->layer\_param\_.spp\_param();
53. num\_ = bottom[0]->num();//batch size 大小
54. channels\_ = bottom[0]->channels();//特征图个数
55. bottom\_h\_ = bottom[0]->height();//特征图宽高
56. bottom\_w\_ = bottom[0]->width();
57. reshaped\_first\_time\_ = **false**;
58. CHECK\_GT(bottom\_h\_, 0) << "Input dimensions cannot be zero.";
59. CHECK\_GT(bottom\_w\_, 0) << "Input dimensions cannot be zero.";
61. pyramid\_height\_ = spp\_param.pyramid\_height();//金子塔有多少层
62. split\_top\_vec\_.clear();//清空相关数据
63. pooling\_bottom\_vecs\_.clear();
64. pooling\_layers\_.clear();
65. pooling\_top\_vecs\_.clear();
66. pooling\_outputs\_.clear();
67. flatten\_layers\_.clear();
68. flatten\_top\_vecs\_.clear();
69. flatten\_outputs\_.clear();
70. concat\_bottom\_vec\_.clear();
71. //如果金字塔只有一层，那么我们其实是对一整张图片进行pooling，也就是文献所提到的：global pooling
72. **if** (pyramid\_height\_ == 1) {
73. // pooling layer setup
74. LayerParameter pooling\_param = GetPoolingParam(0, bottom\_h\_, bottom\_w\_,spp\_param);
75. pooling\_layers\_.push\_back(shared\_ptr<PoolingLayer<Dtype> > (**new** PoolingLayer<Dtype>(pooling\_param)));
76. pooling\_layers\_[0]->SetUp(bottom, top);
77. **return**;
78. }
79. //这个将用于保存金子塔每一层
80. **for** (**int** i = 0; i < pyramid\_height\_; i++) {
81. split\_top\_vec\_.push\_back(**new** Blob<Dtype>());
82. }
84. // split layer setup
85. LayerParameter split\_param;
86. split\_layer\_.reset(**new** SplitLayer<Dtype>(split\_param));
87. split\_layer\_->SetUp(bottom, split\_top\_vec\_);
89. **for** (**int** i = 0; i < pyramid\_height\_; i++) {
90. // pooling layer input holders setup
91. pooling\_bottom\_vecs\_.push\_back(**new** vector<Blob<Dtype>\*>);
92. pooling\_bottom\_vecs\_[i]->push\_back(split\_top\_vec\_[i]);

95. pooling\_outputs\_.push\_back(**new** Blob<Dtype>());
96. pooling\_top\_vecs\_.push\_back(**new** vector<Blob<Dtype>\*>);
97. pooling\_top\_vecs\_[i]->push\_back(pooling\_outputs\_[i]);
99. // 获取金字塔每一层相关参数
100. LayerParameter pooling\_param = GetPoolingParam(i, bottom\_h\_, bottom\_w\_, spp\_param);
102. pooling\_layers\_.push\_back(shared\_ptr<PoolingLayer<Dtype> > (**new** PoolingLayer<Dtype>(pooling\_param)));
103. pooling\_layers\_[i]->SetUp(\*pooling\_bottom\_vecs\_[i], \*pooling\_top\_vecs\_[i]);
105. //每一层金字塔输出向量
106. flatten\_outputs\_.push\_back(**new** Blob<Dtype>());
107. flatten\_top\_vecs\_.push\_back(**new** vector<Blob<Dtype>\*>);
108. flatten\_top\_vecs\_[i]->push\_back(flatten\_outputs\_[i]);
110. // flatten layer setup
111. LayerParameter flatten\_param;
112. flatten\_layers\_.push\_back(**new** FlattenLayer<Dtype>(flatten\_param));
113. flatten\_layers\_[i]->SetUp(\*pooling\_top\_vecs\_[i], \*flatten\_top\_vecs\_[i]);
115. // concat layer input holders setup
116. concat\_bottom\_vec\_.push\_back(flatten\_outputs\_[i]);
117. }
119. // 把所有金字塔层的输出，串联成一个特征向量
120. LayerParameter concat\_param;
121. concat\_layer\_.reset(**new** ConcatLayer<Dtype>(concat\_param));
122. concat\_layer\_->SetUp(concat\_bottom\_vec\_, top);
123. }

函数GetPoolingParam是我们需要细读的函数，里面设置了金子塔每一层窗口大小的计算，其它的函数就不贴了，对caffe底层实现感兴趣的，可以自己慢慢细读。

**四、算法应用之物体检测**

在SPP-Net还没出来之前，物体检测效果最牛逼的应该是RCNN算法了，下面跟大家简单讲一下R-CNN的总算法流程，简单回顾一下：

1、首先通过选择性搜索，对待检测的图片进行搜索出2000个候选窗口。

2、把这2k个候选窗口的图片都缩放到227\*227，然后分别输入CNN中，每个候选窗台提取出一个特征向量，也就是说利用CNN进行提取特征向量。

3、把上面每个候选窗口的对应特征向量，利用SVM算法进行分类识别。

可以看到R-CNN计算量肯定很大，因为2k个候选窗口都要输入到CNN中，分别进行特征提取，计算量肯定不是一般的大。

OK，接着回归正题，如何利用SPP-Net进行物体检测识别？具体算法的大体流程如下：

1、首先通过选择性搜索，对待检测的图片进行搜索出2000个候选窗口。这一步和R-CNN一样。

2、特征提取阶段。这一步就是和R-CNN最大的区别了，同样是用卷积神经网络进行特征提取，但是SPP-Net用的是金字塔池化。这一步骤的具体操作如下：把整张待检测的图片，输入CNN中，进行一次性特征提取，得到feature maps，然后在feature maps中找到各个候选框的区域，再对各个候选框采用金字塔空间池化，提取出固定长度的特征向量。而R-CNN输入的是每个候选框，然后在进入CNN，因为SPP-Net只需要一次对整张图片进行特征提取，速度是大大地快啊。江湖传说可一个提高100倍的速度，因为R-CNN就相当于遍历一个CNN两千次，而SPP-Net只需要遍历1次。

3、最后一步也是和R-CNN一样，采用SVM算法进行特征向量分类识别。

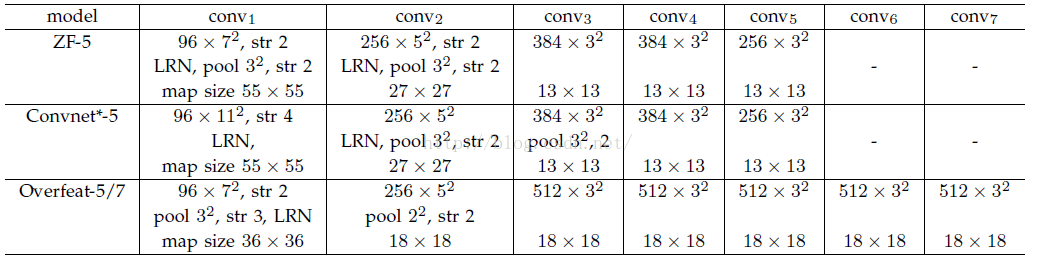
算法细节说明：看完上面的步骤二，我们会有一个疑问，那就是如何在feature maps中找到原始图片中候选框的对应区域？因为候选框是通过一整张原图片进行检测得到的，而feature maps的大小和原始图片的大小是不同的，feature maps是经过原始图片卷积、下采样等一系列操作后得到的。那么我们要如何在feature maps中找到对应的区域呢？这个答案可以在文献中的最后面附录中找到答案：APPENDIX A：Mapping a Window to Feature Maps。这个作者直接给出了一个很方便我们计算的公式：假设(x’,y’)表示特征图上的坐标点，坐标点(x,y)表示原输入图片上的点，那么它们之间有如下转换关系：

(x,y)=(S\*x’,S\*y’)

其中S的就是CNN中所有的strides的乘积。比如paper所用的ZF-5：

S=2\*2\*2\*2=16

而对于Overfeat-5/7就是S=12，这个可以看一下下面的表格：



需要注意的是Strides包含了池化、卷积的stride。自己计算一下Overfeat-5/7(前5层)是不是等于12。

反过来，我们希望通过(x,y)坐标求解(x’,y’)，那么计算公式如下：

http://img.blog.csdn.net/20151227222354752?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

因此我们输入原图片检测到的windows，可以得到每个矩形候选框的四个角点，然后我们再根据公式：

Left、Top:

http://img.blog.csdn.net/20151227222424156?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

Right、Bottom：

http://img.blog.csdn.net/20151227222501283?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

参考文献：

1、[https://github.com/BVLC/caffe](https://github.com/BVLC/caffe" \t "_blank)

2、《Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition》

3、[http://research.microsoft.com/en-us/um/people/kahe/eccv14sppnet/index.html](http://research.microsoft.com/en-us/um/people/kahe/eccv14sppnet/index.html" \t "_blank)

4、[http://caffe.berkeleyvision.org/](http://caffe.berkeleyvision.org/" \t "_blank)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*作者：hjimce   时间：2015.12.5  联系QQ：1393852684   地址：<http://blog.csdn.net/hjimce>   原创文章，版权所有，转载请保留本行信息