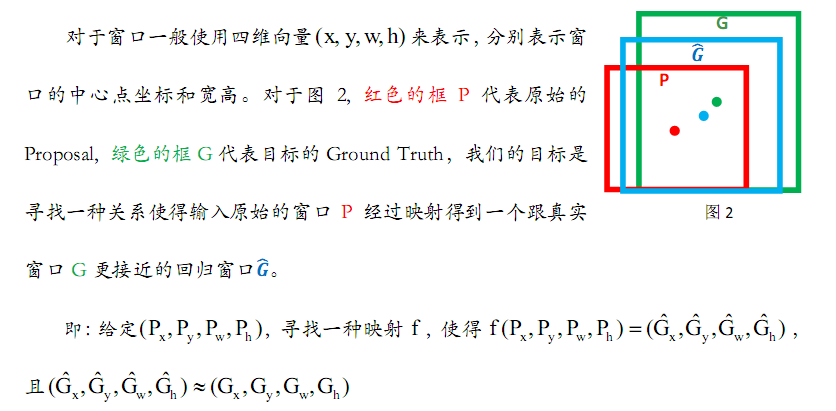
**Bounding box Regression分析**

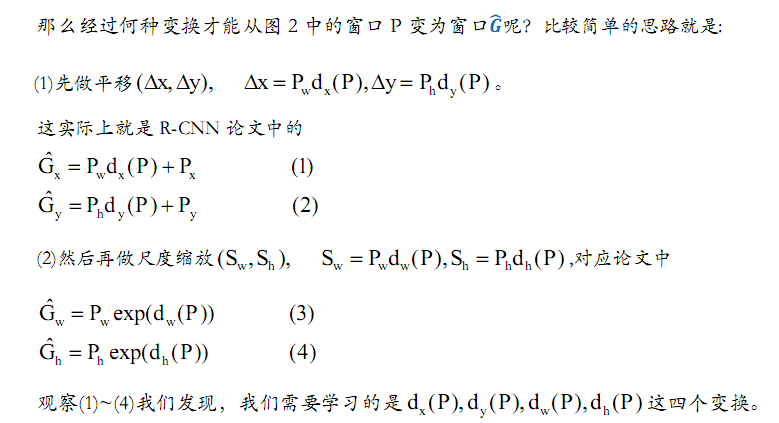
1. **问题的数学表达，回归的对象是什么？**

我们一般最开始都有一个初始的BBox，但是这只是个粗略的BBox。比如RCNN，fastrcnn用Selective Search方法生成的一系列Proposals，faster-rcnn的anchors对应的boxes。这些都是粗略的框。我们最终要得到精确的位置框，所以需要用回归的方法精修BBox。

因此，我们的输入是初始的BBox，需要找到一个映射把初始的BBox精修到最后Ground Truth。对应的数学描述如下：

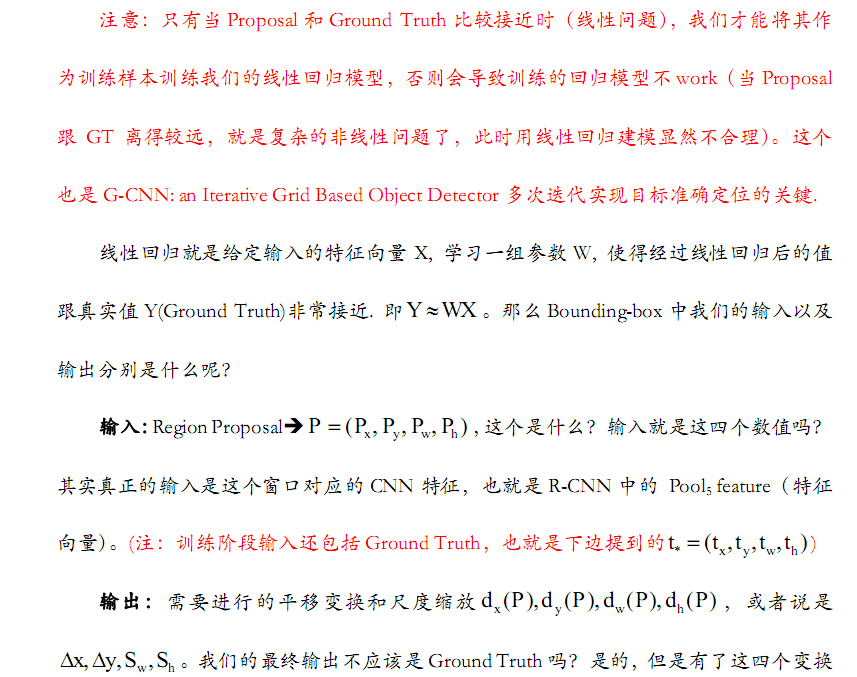


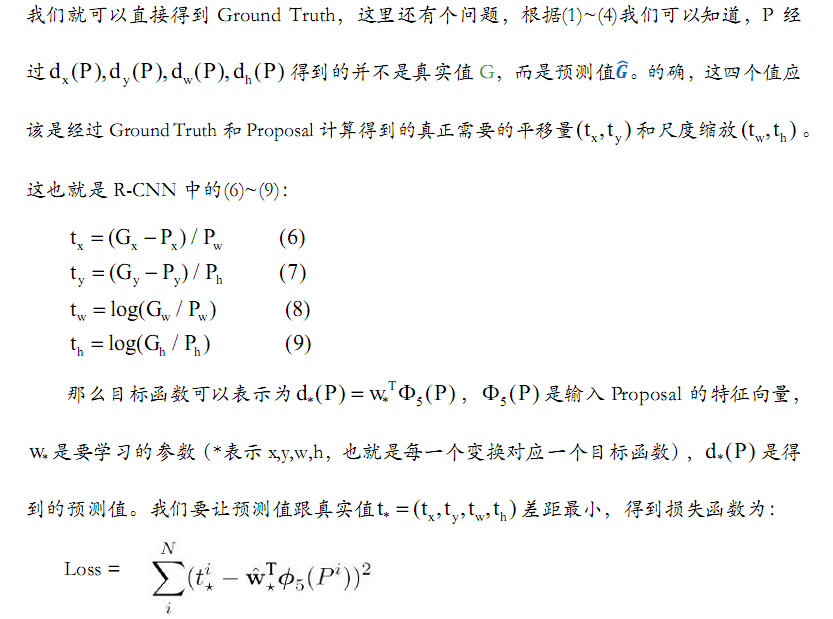
1. **如何进行回归**

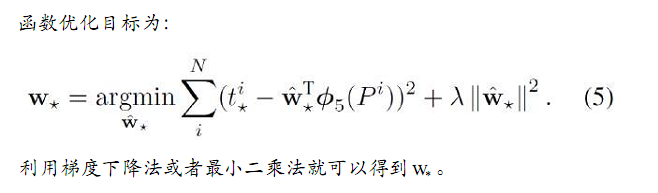


下一步的问题就是设计算法得到这四个映射。

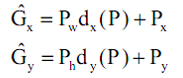
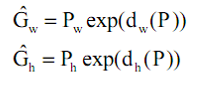
当输入的Proposals与Ground Truth相差较小时，（RCNN中设置的是IOU>0.6），可以认为这种变化是一种线性变化，因此可以用线性回归对该问题建模，从而微调窗口。







这样以来，我们回归的结果是原始Proposals需要微调的dx,dy,dw,dh。最后再使用下面的公式也就是上面的公式（1）~（4）得到最后的回归框。

1. **rcnn中的实现**

rcnn使用Selective Search的方法从一张图像生成约2000-3000个候选区域。基本思路如下：

- 使用一种过分割手段，将图像分割成小区域

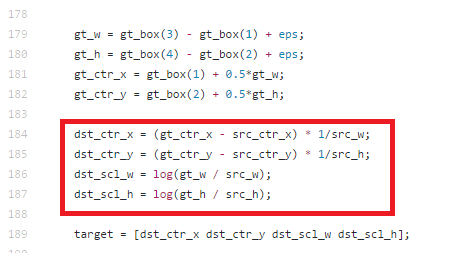
- 查看现有小区域，合并可能性最高的两个区域。

重复直到整张图像合并成一个区域位置

- 输出所有曾经存在过的区域，所谓候选区域

Rcnn中候选区域生成和后续步骤相对独立，也就是在rcnn网络中，提前用Selective Search的方法已经得到了上面将的候选框，即Px，Py，Pw，Ph。RCNN通过线性回归器进行每个候选框的精修工作，即回归器的输入为深度网络pool5层的4096维特征（这里使用的是alexnet），输出为xy方向的缩放和平移,即dx,dy,dh,dw。

由于rcnn比较简单，这里不再讲述，这列出一张计算t的实例代码。



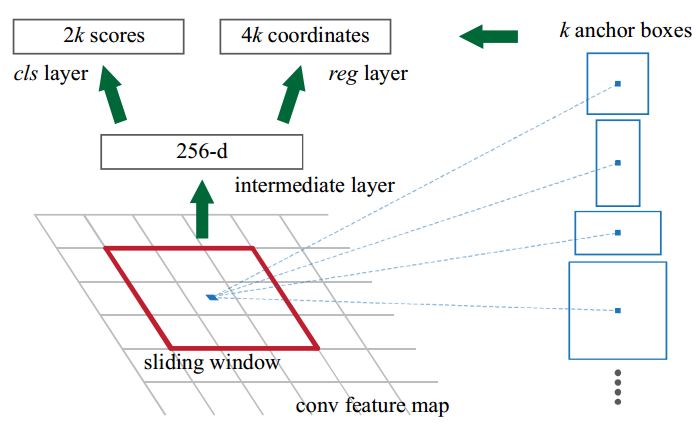
下面会重点讲述faster rcnn。

1. **Faster rcnn中的实现**

Faster –rcnn中Bbox回归的实现。

1.网络的RPN

RPN的核心思想是使用卷积神经网络直接产生region proposal，使用的方法本质上就是滑动窗口。Region ProposalNetwork(RPN)如下图：



RPN网络结构图如上所示（ZF模型:256维），假设给定600\*1000的输入图像，经过卷积操作得到最后一层的卷积feature map（大小约为40\*60），最后一层卷积层共有256个feature map。

在这个特征图上使用3\*3的卷积核（滑动窗口）与特征图进行卷积，那么这个3\*3的区域卷积后可以获得一个256维的特征向量。因为这个3\*3的区域上，每一个特征图上得到一个1维向量，256个特性图即可得到256维特征向量。

3\*3滑窗中心点位置，对应预测输入图像3种尺度（128,256,512），3种长宽比（1:1,1:2,2:1）的regionproposal，这种映射的机制称为anchor，产生了k=9个anchor。即每个3\*3区域可以产生9个region proposal。所以对于这个40\*60的feature map，总共有40\*60个anchor，约20000(40\*60\*9)个proposal，也就是预测20000个region proposal。

下面是RPN回归的网络结构。

#========= RPN ============

layer {

name: "rpn\_conv/3x3"

type: "Convolution"

bottom: "conv5\_3"

top: "rpn/output"

param { lr\_mult: 1.0 }

param { lr\_mult: 2.0 }

convolution\_param {

num\_output: 512

kernel\_size: 3 pad: 1 stride: 1

weight\_filler { type: "gaussian" std: 0.01 }

bias\_filler { type: "constant" value: 0 }

}

}

layer {

name: "rpn\_relu/3x3"

type: "ReLU"

bottom: "rpn/output"

top: "rpn/output"

}

。。。。。。

layer {

name: "rpn\_bbox\_pred"

type: "Convolution"

bottom: "rpn/output"

top: "rpn\_bbox\_pred"

param { lr\_mult: 1.0 }

param { lr\_mult: 2.0 }

convolution\_param {

num\_output: 36 # 4 \* 9(anchors)

kernel\_size: 1 pad: 0 stride: 1

weight\_filler { type: "gaussian" std: 0.01 }

bias\_filler { type: "constant" value: 0 }

}

}

。。。。。。

layer {

name: 'rpn-data'

type: 'Python'

bottom: 'rpn\_cls\_score'

bottom: 'gt\_boxes'

bottom: 'im\_info'

bottom: 'data'

top: 'rpn\_labels'

top: 'rpn\_bbox\_targets'

top: 'rpn\_bbox\_inside\_weights'

top: 'rpn\_bbox\_outside\_weights'

python\_param {

module: 'rpn.anchor\_target\_layer'

layer: 'AnchorTargetLayer'

param\_str: "'feat\_stride': 16"

}

}

。。。。。。

layer {

name: "rpn\_loss\_bbox"

type: "SmoothL1Loss"

bottom: "rpn\_bbox\_pred"

bottom: "rpn\_bbox\_targets"

bottom: 'rpn\_bbox\_inside\_weights'

bottom: 'rpn\_bbox\_outside\_weights'

top: "rpn\_loss\_bbox"

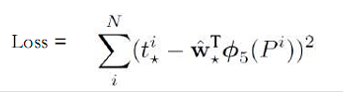
loss\_weight: 1

smooth\_l1\_loss\_param { sigma: 3.0 }

}

rpn\_bbox\_pred层用卷积层，实现了线性回归，输出了36个通道的特征图。这里36表示每个anchors有9个boxes，每个box需要回归dx,dy,dw,dh四个参数。所以每个anchors一共对应4\*9=36个值。如果输入40\*60的feature map，总共有约40\*60个anchors。

rpn\_loss\_bbox层计算损失函数，也就是上面分析的损失函数，即



只不过这里计算的是SmoothL1Loss，不求平方，另外第二项就是这里rpn\_bbox\_pred的输出，即rpn\_bbox\_pred。第一项就是真实值，即rpn\_bbox\_targets。

下面看看rpn\_bbox\_targets是怎么计算的。

在AnchorTargetLayer 层中完成了rpn\_bbox\_targets的计算。

AnchorTargetLayer位置在 py-faster-rcnn/lib/rpn中。

AnchorTargetLayer类中forward函数中计算了bbox\_targets。

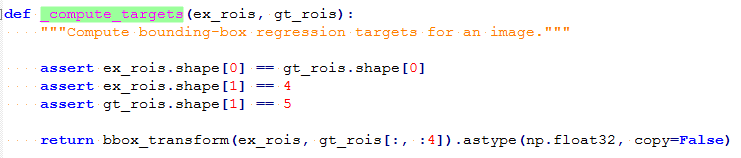


可见bbox\_targets是用anchors和gt\_box计算出来的。

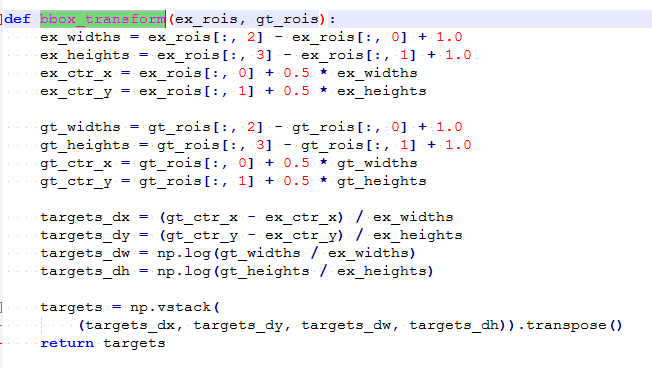
下面分两步讲解，先讲一下\_compute\_targets的实现再讲anchors是怎么生成的。

（1）\_compute\_targets的实现。

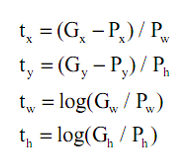
AnchorTargetLayer类中实现了\_compute\_targets函数中。



bbox\_transform函数位于py-faster-rcnn/lib/fast-rcnn/bbox\_transform.py文件中。

****

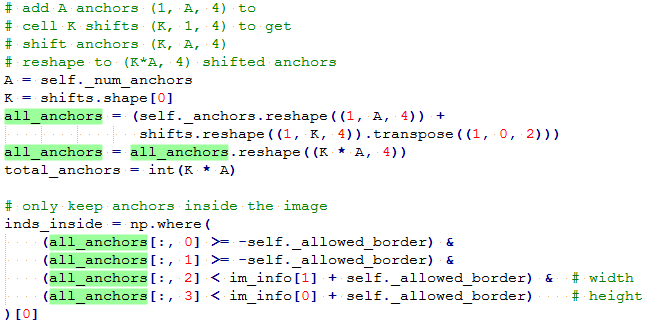
可见，bbox\_transform实现的功能，实际就是上面我们计算t的过程，即



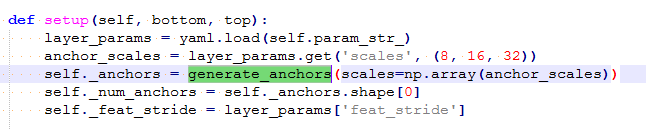
这里的Px,Py,Ph,Pw就是上面计算的anchors，Gx,Gy,Gh,Gw就是上面gt\_box，即我们提前标注的边界框。

（2）anchors怎么计算

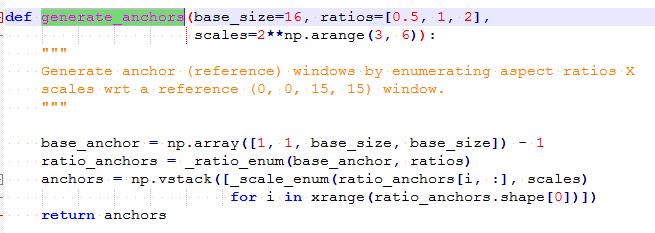
下面是AnchorTargetLayer类中forward函数中用anchors的地方。



可是anchors怎么创建的呢？AnchorTargetLayer类中setup函数实现了这个功能。



generate\_anchors位于py-faster-rcnn/lib/rpn/generate\_anchors.py中。



可见，generate\_anchors函数中定义的anchors的scale为2^3,2^4,2^5,即[8,16,32]，由于\_ratio\_enum函数会将scale乘以base\_size（16），即对应的预测的尺度变为（128,256,512）。另外Ratios为[0.5,1,2]。这里，和之前讲述RPN中所说的，对应预测输入图像3种尺度（128,256,512），3种长宽比（1:1,1:2,2:1）的region proposal是一致的。

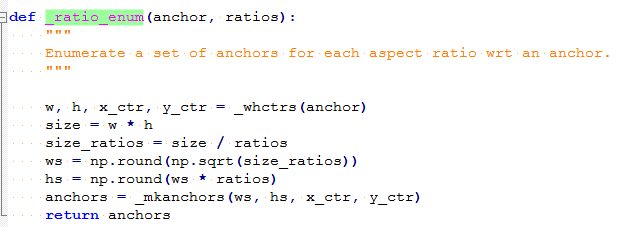
base\_anchor对应一个box的x,y,w,h。即一个region proposal的x,y,w,h。

ratio\_anchors变量是对应3个不同比例的boxes的x,y,w,h；

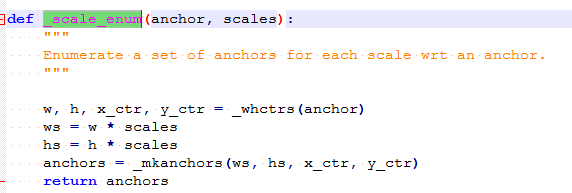
\_scale\_enum（ratio\_anchors[i,:], scales）中scales是个二维矩阵，矩阵为3行，每一行对应一个box的x,y,w,h，ratio\_anchors[i,:]也是二维矩阵，矩阵为3行，每一行对应一个box的x,y,w,h。\_scale\_enum()返回二维矩阵，矩阵为9行，每一行对应一个box的x,y,w,h。即9个不同尺寸，不同rate的box的x,y,w,h。

anchors为特征图所有位置对应的box的x,y,w,h。每个位置对应9个不同尺寸，不同rate的box的x,y,w,h。

\_ratio\_enum函数定义如下：



\_scale\_enum函数定义如下：



1. **SSD中的回归**
2. Default Box 的生成

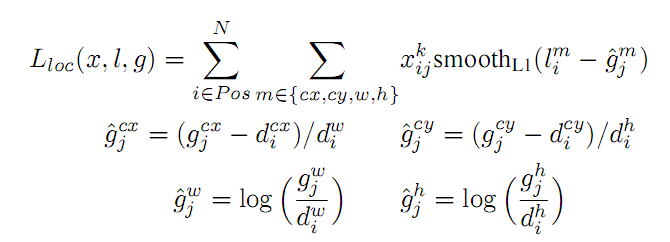
SSD利用不同层的 feature map 来模仿学习不同尺度下物体的检测。假定使用 m 个不同层的feature map 来做预测，最底层的 feature map 的 scale 值 为0.2，最高层的为 0.95，其他层通过下面公式计算得到  
 使用不同的 ratio值,即 为{1，2，3，, } 计算 default box 的宽度和高度：

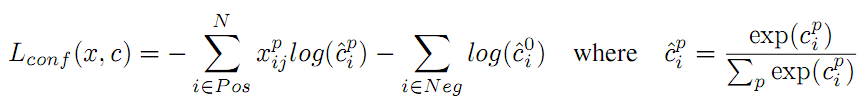
另外对于 ratio = 1 的情况，额外再指定 scale 为

这样，也就是总共有 6 中不同的 default box。每个 default box的中心位置设置成其中 i，j 分别表示为在大小为的 feature map上点的位置，即i,j的取值范围为0到。如在8\*8的feature map中，i为1，j为2，表示第1行，第2列的的位置。（行列从0开始计算）。这样以来，用于预测的 feature map 上的每个点都对应有 6 个不同的 default box。

1. SSD训练的目标函数，

和常见的 Object Detection 的方法目标函数相同，分为两部分：计算相应的 default box 与目标类别的 score(置信度)，以及相应的回归结果（位置回归）。置信度是采用 Softmax Loss，位置回归则是采用 Smooth L1 loss。

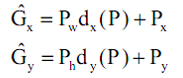
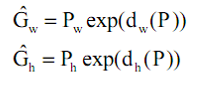




这里就是前面讲RCNN中讲到的t，g就是打标签的值，d是proposal的坐标，即Default Box的坐标。这里就不在详细分析代码了。

1. **mtcnn中的回归**

这里只讲述mtcnn从回归的数据中如果恢复出最终的bbox。因为回归出来的4个数值是dx,dy,dh,dw。如何把这些转换成x,y,w,h。即使用下面的公式得到最后的回归框。

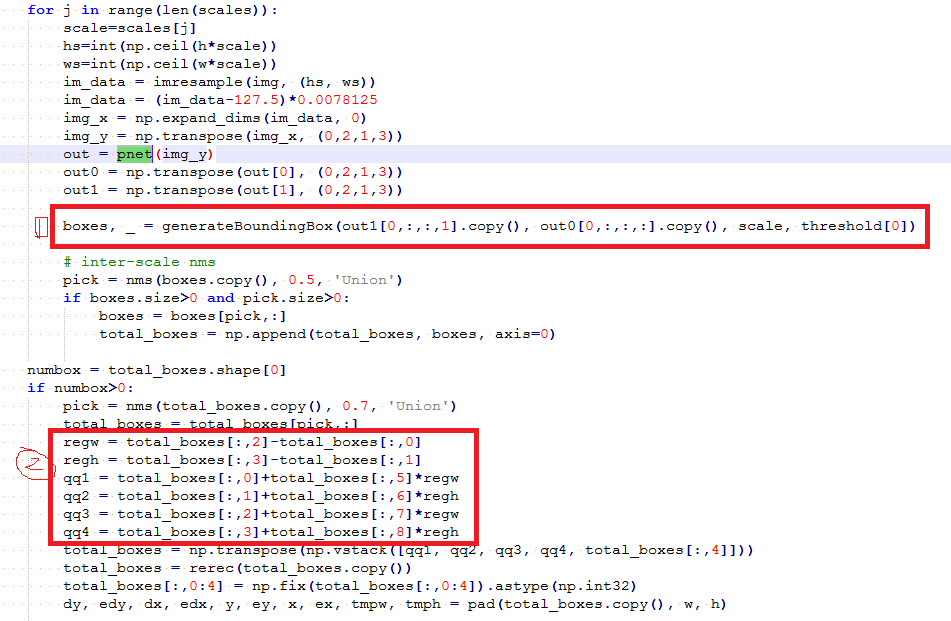
 

下面以P网络的输出举例说明。

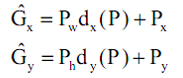
P网络的输出是out。

out第一行所有待选框回归出来的dx1,dy1, dx2,dy2,

out第二行是所有待选框的概率值。



1. 使用generateBoundingBox函数，将预测的概率的大于阈值的候选框的回归值挑选出来。
2. 用下式计算出来Bbox左上角的x,y,和右下角的x,y。这里的total\_boxes[:,5]~ total\_boxes[:,8]对应着回归出来的dx,dy。total\_boxes[:,0]~ total\_boxes[:,4]对应着原始候选框的左上角和右下角坐标。即下面的Px，Py。



下面是generateBoundingBox函数.

stride=2，表示候选框右下角的坐标就是左上角坐标的的2倍。

cellsize=12，表示选框右下角的坐标还需要平移12个单位。

P网络在训练时生成候选框的规则也应该是上面的方式。预测时，候选框的生成方式和训练是生成的候选框保持一致。

由于mtcnn在候选框生成时没有像faster-rcnn，一个anchors会生成不同尺度，不同rate的的候选框，所以mtcnn会一开始先生成图像的金字塔，生成不同尺度的图片，这样以来，最后的候选框就相当于有了不同尺度的候选框。

q1是预测的概率的大于阈值的候选框的左上角坐标（除以scale，转换为在原图中的坐标）。

q2是预测的概率的大于阈值的候选框的右下角坐标（除以scale，转换为在原图中的坐标）。

reg是对应的候选框回归的dx1,dy1, dx2,dy2。

