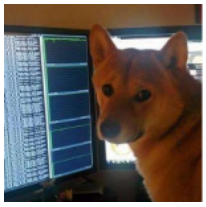
个人资料



潜伏在代码中

+ 关注 发私信

访问： 38879次
积分： 589
等级： BLOG 3
排名： 千里之外

原创： 10篇 转载： 0篇
译文： 0篇 评论： 70条

原 CNN目标检测与分割（一）：Faster RCNN详解

2017-03-14 16:40 12492人阅读 评论(29) 收藏 举报

分类：
CNN图目标检测与分割（2）

↑↑↑↑**目录在这里**↑↑↑↑

Faster RCNN github : https://github.com/rbgirshick/py-faster-rcnn

Faster RCNN paper : https://arxiv.org/abs/1506.01497

Bound box regression详解 :

http://download.csdn.net/download/zy1034092330/9940097（来源：王斌_ICT）

经过RCNN和Fast RCNN的积淀，Ross B. Girshick在2016年提出了新的Faster RCNN，在结构上，Faster RCN已经将特征抽取(feature extraction)，proposal提取，bounding box regression(rect refine)，classification都整合在了一个网络中，使得综合性能有较大提高，在检测速度方面尤为明显。

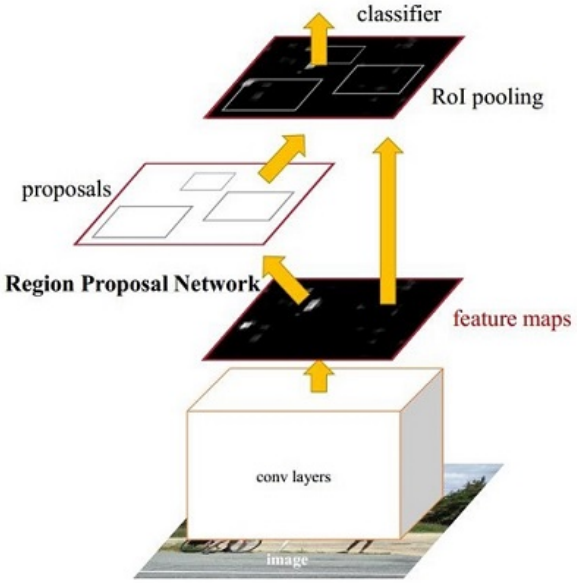


图1 Faster CNN基本结构（来自原论文）

依作者看来，如图1，Faster RCNN其实可以分为4个主要内容：

1. Conv layers。作为一种CNN网络目标检测方法，Faster RCNN首先使用一组基础的conv+relu+pooling层提取image的feature maps。该feature maps被共享用于后续RPN层和全连接层。

2. Region Proposal Networks。RPN网络用于生成region proposals。该层通过softmax判断anchors属于foreground或者background，再利用bounding box regression修正anchors获得精确的proposals。
3. Roi Pooling。该层收集输入的feature maps和proposals，综合这些信息后提取proposal feature maps，送入后续全连接层判定目标类别。
4. Classification。利用proposal feature maps计算proposal的类别，同时再次bounding box regression获得检测框最终的精确位置。

所以本文以上述4个内容作为切入点介绍Faster RCNN网络。

图2展示了python版本中的VGG16模型中的faster_rcnn_test.pt的网络结构，可以清晰的看到该网络对于一副任意大小PxQ的图像，首先缩放至固定大小MxN，然后将MxN图像送入网络；而Conv layers中包含了13个conv层+13个relu层+4个pooling层；RPN网络首先经过3x3卷积，再分别生成foreground anchors与bounding box regression偏移量，然后计算出proposals；而Roi Pooling层则利用proposals从feature maps中提取proposal feature送入后续全连接和softmax网络作classification（即分类proposal到底是什么object）。

path:${py-faster-rcnn-root}/models/pascal_voc/VGG16/faster_rcnn_alt_opt/faster_rcnn_test.pt

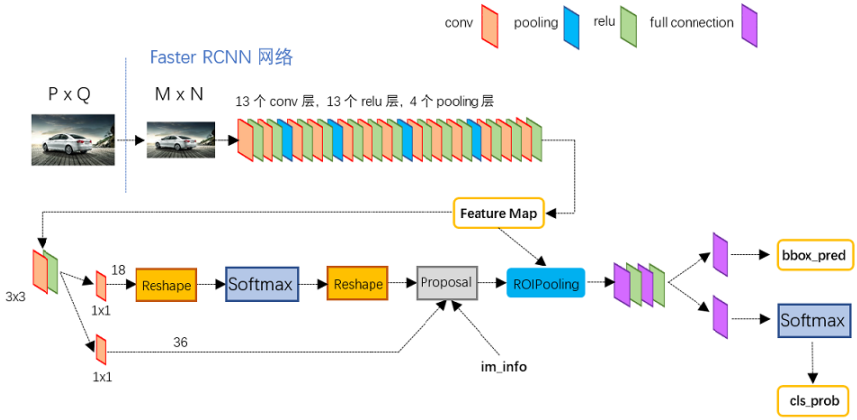


图2 faster_rcnn_test.pt网络结构

1 Conv layers

Conv layers包含了conv，pooling，relu三种层。以python版本中的VGG16模型中的faster_rcnn_test.pt的网络结构为例，如图2，Conv layers部分共有13个conv层，13个relu层，4个pooling层。这里有一个非常容易被忽略但是又无比重要的信息，在Conv layers中：

1. 所有的conv层都是：kernel_size=3，pad=1
2. 所有的pooling层都是：kernel_size=2，stride=2

为何重要？在Faster RCNN Conv layers中对所有的卷积都做了扩边处理（pad=1，即填充一圈0），导致原图变为(M+2)x(N+2)大小，再做3x3卷积后输出MxN。正是这种设置，导致Conv layers中的conv层不改变输入和输出矩阵大小。如图3：

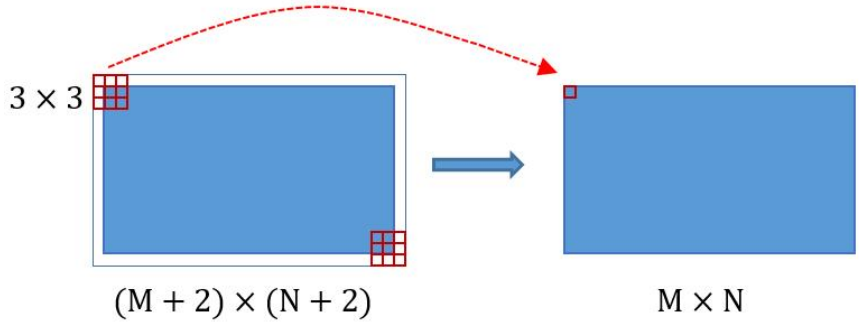


图3

类似的是，Conv layers中的pooling层kernel_size=2，stride=2。这样每个经过pooling层的MxN矩阵，都会变为(M/2)*(N/2)大小。综上所述，在整个Conv layers中，conv和relu层不改变输入输出大小，只有pooling层使输出长宽都变为输入的1/2。

那么，一个MxN大小的矩阵经过Conv layers固定变为(M/16)x(N/16)！这样Conv layers生成的featuure map中都可以和原图对应起来。 4个pooling层

2 Region Proposal Networks(RPN)

经典的检测方法生成检测框都非常耗时，如OpenCV adaboost使用滑动窗口+图像金字塔生成检测框；或如RCNN使用SS(Selective Search)方法生成检测框。而Faster RCNN则抛弃了传统的滑动窗口和SS方法，直接使用RPN生成检测框，这也是Faster RCNN的巨大优势，能极大提升检测框的生成速度。

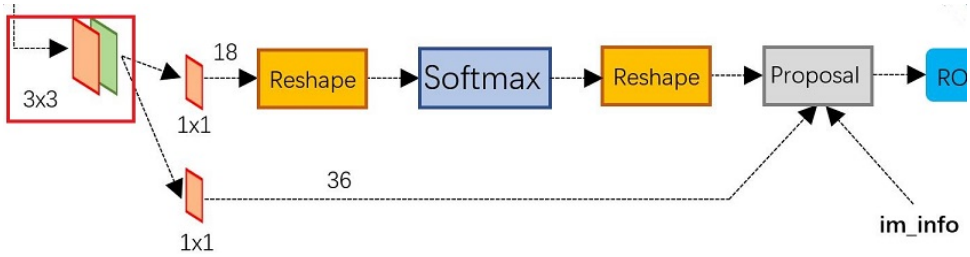


图4 RPN网络结构

上图4展示了RPN网络的具体结构。可以看到RPN网络实际分为2条线，上面一条通过softmax分类anchors获得foreground和background（检测目标是foreground），下面一条用于计算对于anchors的bounding box regression偏移量，以获得精确的proposal。而最后的Proposal层则负责综合foreground anchors和bounding box regression偏移量获取proposals，同时剔除太小和超出边界的proposals。其实整个网络到了Proposal Layer这里，就完成了相当于目标定位的功能。

2.1 多通道图像卷积基础知识介绍

在介绍RPN前，还要多解释几句基础知识，已经懂的看官老爷跳过就好。

1. 对于单通道图像+单卷积核做卷积，第一章中的图3已经展示了；
2. 对于多通道图像+多卷积核做卷积，计算方式如下：

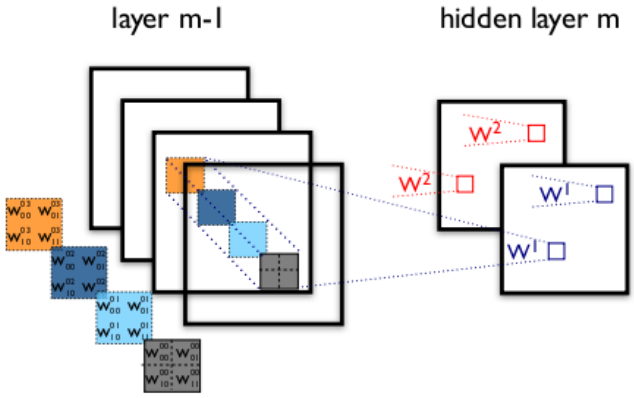


图5 多通道+多卷积核做卷积示意图（摘自Theano教程）

如图5，输入图像layer m-1有4个通道，同时有2个卷积核w1和w2。对于卷积核w1，先在输入图像4个通道分别作卷积，再将4个通道结果加起来得到w1的卷积输出；卷积核w2类似。所以对于某个卷积层，无论输入图像有多少个通道，输出图像通道数总是等于卷积核数量！

对多通道图像做1x1卷积，其实就是将输入图像于每个通道乘以卷积系数后加在一起，即相当于把原图像中本来各个独立的通道“联通”在了一起。

2.2 anchors

提到RPN网络，就不能不说anchors。所谓anchors，实际上就是一组由rpn/generate_anchors.py生成的矩形。直接运行作者demo中的generate_anchors.py可以得到以下输出：

```
[python]
[[ -84.  -40.   99.   55.]
 [-176.  -88.  191.  103.]
 [-360. -184.  375.  199.]
 [ -56.  -56.   71.   71.]
 [-120. -120.  135.  135.]
 [-248. -248.  263.  263.]
 [ -36.  -80.   51.   95.]
 [ -80. -168.   95.  183.]
 [-168. -344.  183.  359.]]
```

其中每行的4个值[x1,y1,x2,y2]代表矩形左上和右下角点坐标。9个矩形共有3种形状，长宽比为大约为：width:height = [1:1, 1:2, 2:1]三种，如图6。实际上通过anchors就引入了检测中常用到的多尺度方法。

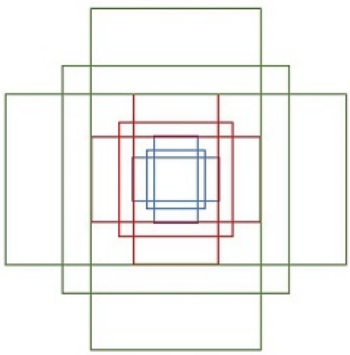


图6 anchors示意图

注：关于上面的anchors size，其实是根据检测图像设置的。在python demo中，会把任意大小的输入图像reshape成800x600（即图2中的M=800，N=600）。再回头来看anchors的大小，anchors中长宽1:2中最大为352x704，长宽2:1中最大736x384，基本是cover了800x600的各个尺度和形状。

那么这9个anchors是做什么的呢？借用Faster RCNN论文中的原图，如图7，遍历Conv layers计算获得的feature maps，为每一个点都配备这9种anchors作为初始的检测框。这样做获得检测框很不准确，不用担心，后面还有2次bounding box regression可以修正检测框位置。

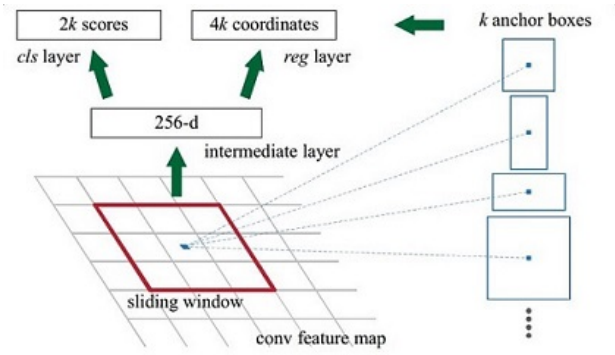


图7

解释一下上面这张图的数字。

1. 在原文中使用的是ZF model中，其Conv Layers中最后的conv5层num_output=256，对应生成256张特征图，所以相当于feature map每个点都是256-d
2. 在conv5之后，做了rpn_conv/3x3卷积且num_output=256，相当于每个点又融合了周围3x3的空间信息（猜测这样做也许更鲁棒？反正我没测试），同时256-d不变（如图4和图7中的红框）
3. 假设在conv5 feature map中每个点上有k个anchor（默认k=9），而每个anhcor要分foreground和background，所以每个点由256d feature转化为cls=2k scores；而每个anchor都有[x, y, w, h]对应4个偏移量，所以reg=4k coordinates
4. 补充一点，全部anchors拿去训练太多了，训练程序会在合适的anchors中**随机**选取128个postive anchors+128个negative anchors进行训练（什么是合适的anchors下文5.1有解释）

注意，在本文讲解中使用的VGG conv5 num_output=512，所以是512d，其他类似.....

2.3 softmax判定foreground与background

一副MxN大小的矩阵送入Faster RCNN网络后，到RPN网络变为(M/16)x(N/16)，不妨设W=M/16，H=N/16。在进入reshape与softmax之前，先做了1x1卷积，如图8：

图8 RPN中判定fg/bg网络结构

该1x1卷积的caffe prototxt定义如下：

```
[cpp]
layer {
  name: "rpn_cls_score"
  type: "Convolution"
  bottom: "rpn/output"
  top: "rpn_cls_score"
  convolution_param {
    num_output: 18   # 2(bg/fg) * 9(anchors)
    kernel_size: 1 pad: 0 stride: 1
  }
}
```

关闭

可以看到其num_output=18，也就是经过该卷积的输出图像为WxHx18大小（注意第二章开头提到的卷积计算方式）。这也就刚好对应了feature maps每一个点都有9个anchors，同时每个anchors又有可能是foreground和background，所有这些信息都保存WxHx(9x2)大小的矩阵。为何这样做？后面接softmax分类获得foreground anchors，也就相当于初步提取了检测目标候选区域box（一般认为目标在foreground anchors中）。

那么为何要在softmax前后都接一个reshape layer？其实只是为了便于softmax分类，至于具体原因这就要从caffe的实现形式说起了。在caffe基本数据结构blob中以如下形式保存数据：

blob=[batch_size, channel，height，width]

对应至上面的保存bg/fg anchors的矩阵，其在caffe blob中的存储形式为[1, 2*9, H, W]。而在softmax分类时需要进行fg/bg二分类，所以reshape layer会将其变为[1, 2, 9*H, W]大小，即单独“腾空”出来一个维度以便softmax分类，之后再reshape回复原状。贴一段caffe softmax_loss_layer.cpp的reshape函数的解释，非常精辟：

```cpp
"Number of labels must match number of predictions; "
"e.g., if softmax axis == 1 and prediction shape is (N, C, H, W), "
"label count (number of labels) must be N*H*W, "
"with integer values in {0, 1, ..., C-1}.";
```

综上所述，RPN网络中利用anchors和softmax初步提取出foreground anchors作为候选区域。

2.4 bounding box regression原理

介绍bounding box regression数学模型及原理。如图9所示绿色框为飞机的Ground Truth(GT)，红色为提取的foreground anchors，那么即便红色的框被分类器识别为飞机，但是由于红色的框定位不准，这张图相当于没有正确的检测出飞机。所以我们希望采用一种方法对红色的框进行微调，使得foreground anchors和GT更加接近。

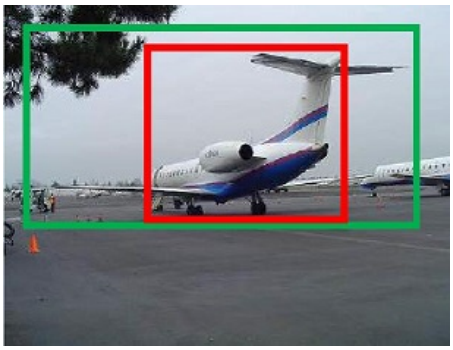


图9

对于窗口一般使用四维向量(x, y, w, h)表示，分别表示窗口的中心点坐标和宽高。对于图 10，红色的框A代表原始的Foreground Anchors，绿色的框G代表目标的GT，我们的目标是寻找一种关系，使得输入原始的anchor A经过映射得到一个跟真实窗口G更接近的回归窗口G'，即：给定anchor $A=(A_x, A_y, A_w, A_h)$，$GT=[G_x, G_y, G_w, G_h]$，寻找一种变换**F**：使得$\mathbf{F}(A_x, A_y, A_w, A_h)=(G'_x, G'_y, G'_w, G'_h)$，其中$(G'_x, G'_y, G'_w, G'_h)\approx(G_x, G_y, G_w, G_h)$。

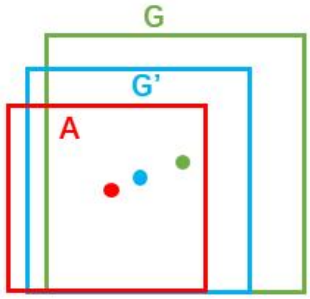


图10

那么经过何种变换**F**才能从图6中的anchor A变为G'呢？ 比较简单的思路就是:

1. 先做平移

$$G'_x = A_w \cdot d_x(A) + A_x$$
$$G'_y = A_h \cdot d_y(A) + A_y$$

2. 再做缩放

$$G'_w = A_w \cdot \exp\left(d_w(A)\right)$$
$$G'_h = A_h \cdot \exp\left(d_h(A)\right)$$

观察上面4个公式发现，需要学习的是$d_x(A)$，$d_y(A)$，$d_w(A)$，$d_h(A)$这四个变换。当输入的anchor A与GT相差较小时，可以认为这种变换是一种线性变换， 那么就可以用线性回归来建模对窗口进行微调（注意，只有当anchors A和GT比较接近时，才能使用线性回归模型，否则就是复杂的非线性问题了）。对应于Faster RCNN原文，平移量(tx, ty)与尺度因子(tw, th)如下：

$$t_{\mathrm{x}} = (x - x_{\mathrm{a}})/w_{\mathrm{a}}, \quad t_{\mathrm{y}} = (y - y_{\mathrm{a}})/h_{\mathrm{a}},$$
$$t_{\mathrm{w}} = \log(w/w_{\mathrm{a}}), \quad t_{\mathrm{h}} = \log(h/h_{\mathrm{a}}),$$

接下来的问题就是如何通过线性回归获得$d_x(A)$，$d_y(A)$，$d_w(A)$，$d_h(A)$了。线性回归就是给定输入的特征向量X, 学习一组参数W, 使得经过线性回归后的值跟真实值Y非常接近，即Y=WX。对于该问题，输入X是一张经过卷积获得的feature map，定义为Φ；同时还有训练传入的GT，即(tx, ty, tw, th)。输出是$d_x(A)$，$d_y(A)$，$d_w(A)$，$d_h(A)$四个变换。那么目标函数可以表示为：

$$d_*(A) = w_*^T \cdot \Phi(\mathrm{A})$$

其中Φ(A)是对应anchor的feature map组成的特征向量，w是需要学习的参数，d(A)是得到的预测值（*表示 x，y，w，h，也就是每一个变换对应一个上述目标函数）。为了让预测值(tx, ty, tw, th)与真实值差距最小，设计损失函数：

$$\mathrm{Loss} = \sum_i^N \left(t_*^i - \hat{w}_*^T \cdot \Phi\left(A^i\right)\right)^2$$

函数优化目标为：

$$w_* = \underset{\hat{w}_*}{\operatorname{argmin}} \sum_i^N \left(t_*^i - \hat{w}_*^T \cdot \Phi\left(A^i\right)\right)^2 + \lambda \|\hat{w}_*\|^2$$

2.5 对proposals进行bounding box regression

在了解bounding box regression后，再回头来看RPN网络第二条线路，如图11。

图11 RPN中的bbox reg

先来看一看上图11中1x1卷积的caffe prototxt定义：

```
[cpp]
layer {
  name: "rpn_bbox_pred"
  type: "Convolution"
  bottom: "rpn/output"
  top: "rpn_bbox_pred"
  convolution_param {
    num_output: 36   # 4 * 9(anchors)
    kernel_size: 1 pad: 0 stride: 1
  }
}
```

可以看到其num_output=36，即经过该卷积输出图像为WxHx36，在caffe blob存储为[1, 36, H, W]，这里相当于feature maps每个点都有9个anchors，每个anchors又都有4个用于回归的$[d_x(A)，d_y(A)，d_w(A)，d_h(A)]$变换量。

2.6 Proposal Layer

Proposal Layer负责综合所有$[d_x(A)，d_y(A)，d_w(A)，d_h(A)]$变换量和foreground anchors，计算出精准的proposal，送入后续RoI Pooling Layer。还是先来看看Proposal Layer的caffe prototxt定义：

```
[cpp]
layer {
  name: 'proposal'
  type: 'Python'
  bottom: 'rpn_cls_prob_reshape'
  bottom: 'rpn_bbox_pred'
  bottom: 'im_info'
  top: 'rois'
  python_param {
    module: 'rpn.proposal_layer'
    layer: 'ProposalLayer'
    param_str: "'feat_stride': 16"
  }
}
```

Proposal Layer有3个输入：fg/bg anchors分类器结果rpn_cls_prob_reshape，对应的bbox reg的$[d_x(A)，d_y(A)，d_w(A)，d_h(A)]$变换量rpn_bbox_pred，以及im_info；另外还有参数feat_stride=16，这和图4是对应的。

首先解释im_info。对于一副任意大小PxQ图像，传入Faster RCNN前首先reshape到固定MxN，im_info=[M, N, scale_factor]则保存了此次缩放的所有信息。然后经过Conv Layers，经过4次pooling变为WxH=(M/16)x(N/16)大小，其中feature_stride=16则保存了该信息，用于计算anchor偏移量。

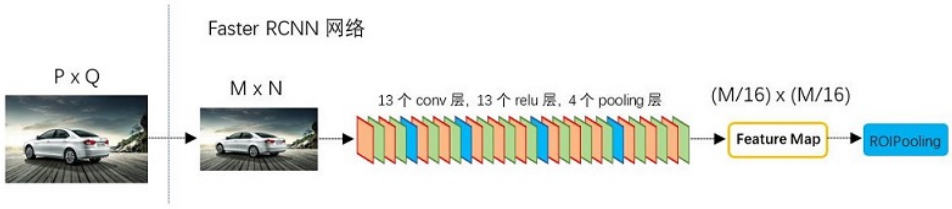


图12

Proposal Layer forward（caffe layer的前传函数）按照以下顺序依次处理：

1. 生成anchors，利用$[d_x(A)，d_y(A)，d_w(A)，d_h(A)]$对所有的anchors做bbox regression回归（这里的anchors生成和训练时完全一致）

2. 按照输入的foreground softmax scores由大到小排序anchors，提取前pre_nms_topN(e.g. 6000)个anchors，即提取修正位置后的foreground anchors。
3. 利用im_info将fg anchors从MxN尺度映射回PxQ原图，判断fg anchors是否大范围超过边界，剔除严重超出边界fg anchors。
4. 进行nms（nonmaximum suppression，非极大值抑制）
5. 再次按照nms后的foreground softmax scores由大到小排序fg anchors，提取前post_nms_topN(e.g. 300)结果作为proposal输出。

之后输出proposal=[x1, y1, x2, y2]，注意，由于在第三步中将anchors映射回原图判断是否超出边界，所以这里输出的proposal是对应MxN输入图像尺度的，这点在后续网络中有用。另外我认为，严格意义上的检测应该到此就结束了，后续部分应该属于识别了~

RPN网络结构就介绍到这里，总结起来就是：

生成anchors -> softmax分类器提取fg anchors -> bbox reg回归fg anchors -> Proposal Layer生成proposals

3 RoI pooling

而RoI Pooling层则负责收集proposal，并计算出proposal feature maps，送入后续网络。从图2中可以看到RoI pooling层有2个输入：

1. 原始的feature maps
2. RPN输出的proposal boxes（大小各不相同）

3.1 为何需要RoI Pooling

先来看一个问题：对于传统的CNN（如AlexNet，VGG），当网络训练好后输入的图像尺寸必须是固定值，同时网络输出也是固定大小的vector or matrix。如果输入图像大小不定，这个问题就变得比较麻烦。有2种解决办法：

1. 从图像中crop一部分传入网络
2. 将图像warp成需要的大小后传入网络

图13 crop与warp破坏图像原有结构信息

两种办法的示意图如图13，可以看到无论采取那种办法都不好，要么crop后破坏了图像的完整结构，要么warp破坏了图像原始形状信息。回忆RPN网络生成的proposals的方法：对foreground anchors进行bound box regression，那么这样获得的proposals也是大小形状各不相同，即也存在上述问题。所以Faster RCNN中提出了RoI Pooling解决这个问题（需要说明，RoI Pooling确实是从SPP发展而来，但是限于篇幅这里略去不讲，有兴趣的读者可以自行查阅相关论文）。

3.2 RoI Pooling原理

分析之前先来看看RoI Pooling Layer的caffe prototxt的定义：

```cpp
layer {
  name: "roi_pool5"
  type: "ROIPooling"
  bottom: "conv5_3"
  bottom: "rois"
  top: "pool5"
  roi_pooling_param {
    pooled_w: 7
    pooled_h: 7
    spatial_scale: 0.0625 # 1/16
  }
}
```

其中有新参数pooled_w=pooled_h=7，另外一个参数spatial_scale=1/16应该能够猜出大概吧。

RoI Pooling layer forward过程：在之前有明确提到：proposal=[x1, y1, x2, y2]是对应MxN尺度的，所以首先使用spatial_scale参数将其映射回(M/16)x(N/16)大小的feature maps尺度（这里来回多次映射，是有点绕）；之后将每个proposal水平和竖直都分为7份，对每一份都进行max pooling处理。这样处理后，即使大小不同的proposal，输出结果都是7x7大小，实现了fixed-length output（固定长度输出）。

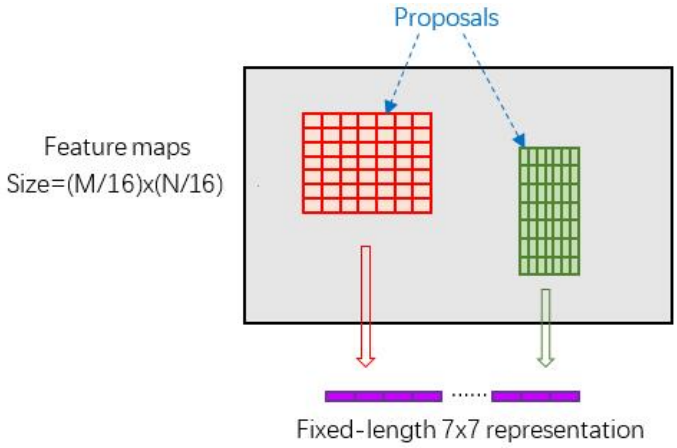


图14 proposal示意图

4 Classification

Classification部分利用已经获得的proposal feature maps，通过full connect层与softmax计算每个proposal具体属于那个类别（如人，车，电视等），输出cls_prob概率向量；同时再次利用bounding box regression获得每个proposal的位置偏移量bbox_pred，用于回归更加精确的目标检测框。Classification部分网络结构如图15。

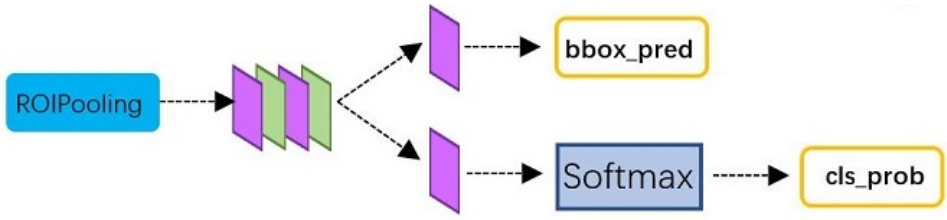


图15 Classification部分网络结构图

从PoI Pooling获取到7x7=49大小的proposal feature maps后，送入后续网络，可以看到做了如下2件事：

1. 通过全连接和softmax对proposals进行分类，这实际上已经是识别的范畴了
2. 再次对proposals进行bounding box regression，获取更高精度的rect box

这里来看看全连接层InnerProduct layers，简单的示意图如图16，

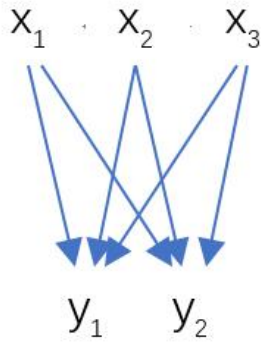


图16 全连接层示意图

其计算公式如下：

$$(x_1 \quad x_2 \quad x_3)\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} + (b_1 \quad b_2) = (y_1 \quad y_2)$$

其中W和bias B都是预先训练好的，即大小是固定的，当然输入X和输出Y也就是固定大小。所以，这也就印证了之前Roi Pooling的必要性。到这里，我想其他内容已经很容易理解，不在赘述了。

5 Faster RCNN训练

Faster CNN的训练，是在已经训练好的model（如VGG_CNN_M_1024，VGG，ZF）的基础上继续进行训练。实际中训练过程分为6个步骤：

1. 在已经训练好的model上，训练RPN网络，对应stage1_rpn_train.pt
2. 利用步骤1中训练好的RPN网络，收集proposals，对应rpn_test.pt
3. 第一次训练Fast RCNN网络，对应stage1_fast_rcnn_train.pt
4. 第二训练RPN网络，对应stage2_rpn_train.pt
5. 再次利用步骤4中训练好的RPN网络，收集proposals，对应rpn_test.pt
6. 第二次训练Fast RCNN网络，对应stage2_fast_rcnn_train.pt

可以看到训练过程类似于一种“迭代”的过程，不过只循环了2次。至于只循环了2次的原因是应为作者提到："A similar alternating training can be run for more iterations, but we have observed negligible improvements"，即循环更多次没有提升了。接下来本章以上述6个步骤讲解训练过程。

5.1 训练RPN网络

在该步骤中，首先读取RBG提供的预训练好的model（本文使用VGG），开始迭代训练。来看看stage1_rpn_train.pt网络结构，如图17。

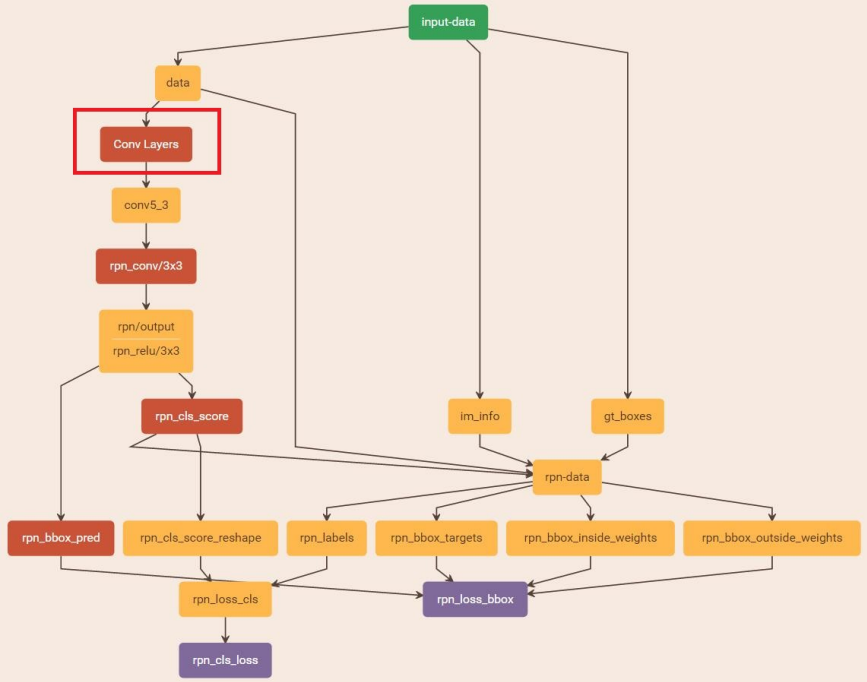


图17 stage1_rpn_train.pt

（考虑图片大小，Conv Layers中所有的层都画在一起了，如红圈所示，后续图都如此处理）

与检测网络类似的是，依然使用Conv Layers提取feature maps。整个网络使用的Loss如下：

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*).$$

上述公式中，i表示anchors index，p_i表示foreground softmax predict概率，p_i^*代表对应的GT predict概率（即当第i个anchor与GT间IoU>0.7，认为是该anchor是foreground，$p_i^*=1$；反之IoU<0.3时，认为是该anchor是background，$p_i^*=0$；至于那些0.3<IoU<0.7的anchor则不参与训练）；t代表predict bounding box，t*代表对应foreground anchor对应的GT box。可以看到，整个Loss分为2部分：

1. cls loss，即rpn_cls_loss层计算的softmax loss，用于分类anchors为forground与background的网络训练
2. reg loss，即rpn_loss_bbox层计算的soomth L1 loss，用于bounding box regression网络训练。注意在该loss中乘了p_i^*，相当于只关心foreground anchors的回归（其实在回归中也完全没必要去关心background）。

由于在实际过程中，N_{cls}和N_{reg}差距过大，用参数λ平衡二者（如$N_{cls}=256$，$N_{reg}=2400$时设置λ=10），使总的网络Loss计算过程中能够均匀考虑2种Loss。这里比较重要是L_{reg}使用的soomth L1 loss，计算公式如下：

$$L_{reg}(t_i, t_i^*) = \sum_{i\epsilon\{x,y,w,h\}} \text{smooth}_{L1}(t_i - t_i^*)$$

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

了解数学原理后，反过来看图17：

1. 在RPN训练阶段，rpn-data（python AnchorTargetLayer）层会按照和test阶段Proposal层完全一样的方式生成Anchors用于训练
2. 对于rpn_loss_cls，输入的rpn_cls_scors_reshape和rpn_labels分别对应p与p*，N_{cls}参数隐含在p与p*的caffe blob的大小中
3. 对于rpn_loss_bbox，输入的rpn_bbox_pred和rpn_bbox_targets分别对应t于t*，rpn_bbox_inside_weigths对应p*，rpn_bbox_outside_weights对应λ，N_{reg}同样隐含在caffe blob大小中

这样，公式与代码就完全对应了。特别需要注意的是，在训练和检测阶段生成和存储anchors的顺序完全一样，这样训练结果才能被用于检测！

5.2 通过训练好的RPN网络收集proposals

在该步骤中，利用之前的RPN网络，获取proposal rois，同时获取foreground softmax probability，如图18，然后将获取的信息保存在python pickle文件中。该网络本质上和检测中的RPN网络一样，没有什么区别。

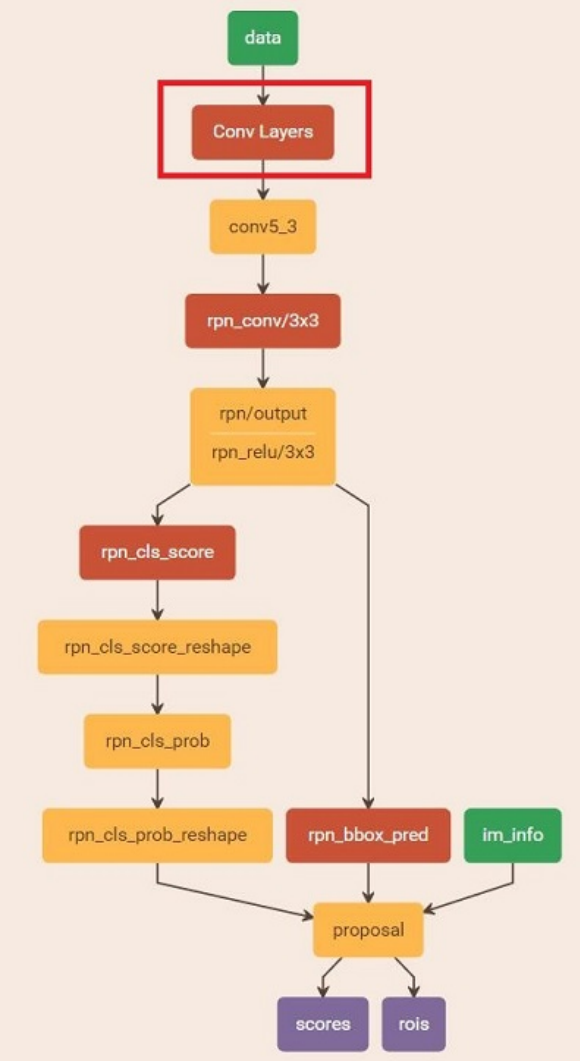


图18 rpn_test.pt

5.3 训练Faster RCNN网络

读取之前保存的pickle文件，获取proposals与foreground probability。从data层输入网络。然后：

1. 将提取的proposals作为rois传入网络，如图19蓝框
2. 将foreground probability作为bbox_inside_weights传入网络，如图19绿框
3. 通过caffe blob大小对比，计算出bbox_outside_weights（即λ），如图19绿框

这样就可以训练最后的识别softmax与最终的bounding regression了，如图19。

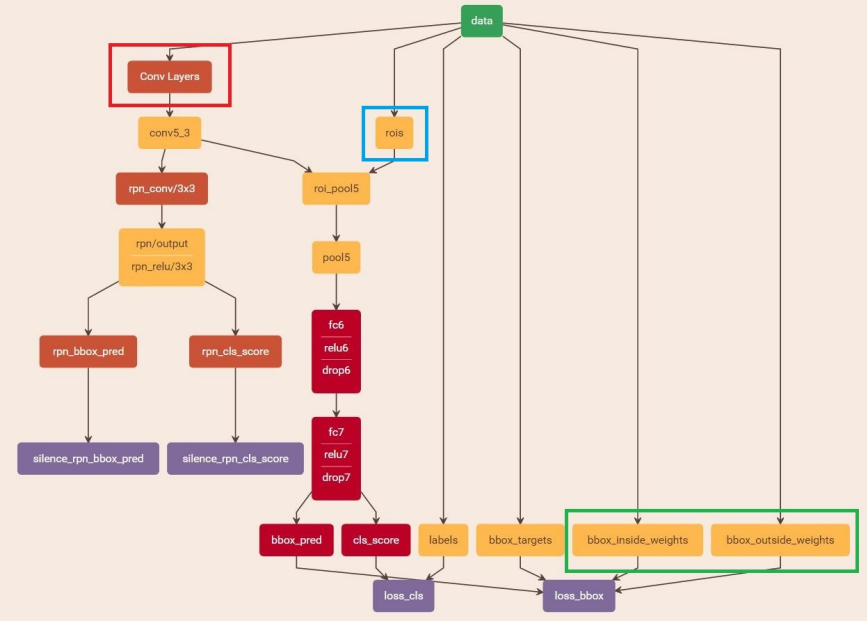


图19 stage1_fast_rcnn_train.pt

之后的训练都是大同小异，不再赘述了。

Faster RCNN还有一种end-to-end的训练方式，可以一次完成train，有兴趣请自己看作者GitHub吧。

PS：我知道你们想问，画图工具：http://ethereon.github.io/netscope/#/editor

Faster RCNN的分析就结束了，之后会缓慢更新YOLO，YOLO V2，SSD，Mask RCNN等内容，敬请期待~

>>>>>>我的YOLO详解点这里<<<<<<

>>>>>>我的SSD详解点这里<<<<<<

大兄弟，你懂的！

- 上一篇 OpenCV中的Haar+Adaboost（七）：分类器训练过程
- 下一篇 CNN目标检测与分割（二）：YOLO

相关文章推荐

- Faster R-CNN学习笔记
- SDCC 2017之大数据技术实战线上峰会
- 各种对象检测论文总结(Object Detection)
- Hadoop大数据从入门到精通知识体系详解
- 使用Faster-Rcnn进行目标检测(实践篇)
- SDCC 2017之区块链技术实战线上峰会
- 自己训练SVM分类器进行HOG行人检测
- C++跨平台开发和ffmpeg，opencv音视频技术
- 自己训练SVM分类器进行HOG行人检测
- 史上最全Android基础知识总结
- CNN目标检测与分割（三）：SSD详解
- 机器学习需要的数学知识
- faster RCNN的Python的画出来loss曲线图
- 【Python】解决使用plt.savefig保存图片时一片...
- 【目标检测】Faster RCNN算法详解
- R-FCN+ResNet-50用自己的数据集训练模型(pyt...

查看评论

IVP_子木 16楼 昨天 21:01发表

楼主，我觉得2.2中的anchors坐标应该是[X,Y,W,H],左上角坐标和高宽哟，不是分别代表左上角和右下角的坐标~

wanghuahua_1003 15楼 3天前 11:33发表

太赞了，终于看的清清楚楚了，博主棒棒哒

slgliwandong 14楼 6天前 17:55发表

博主你好，想问一下，我的原始图片大小是1080*1920大小，目标样本最大有300*300大小，是不是要改一下最大的anchor的生成尺寸，归一化输入的尺寸需要改吗，这两个尺寸在哪里改呢

潜伏在代码中 Re: 4天前 15:06发表

回复slgliwandong：你用1080P的图直接训练？那恐怕要训练很久很久了，不建议这么做。。。
如果确定要这么搞，是要改anchor的大小，保证anchor大小基本要能覆盖你的目标。
具体在哪太久忘了，大概是generate_anchors.py

Saber-alter 13楼 2017-08-23 17:03发表

写的真好，赞

FariverHome 12楼 2017-08-09 11:34发表

楼主的Faster-RCNN讲得非常细致明白。但我有两个小问题没看明白，请教一下：
1. 在im_info上进行ROIpooling会得到每个ROI上的特征图。请问一下，这些特征图是依次传入全连接层进行计算，还是一次性全部传入，得到全部ROI的分类回归结果？找了好久没有找到答案。
2. 进入RPN网络后先进行3x3卷积，之后分别做了两个1x1卷积，之后就可以对anchors进行分类与回归了。但是anchor的尺寸与比例在这个过程中并没有用到，而是在proposal_layer中generate_anchors

函数中才用到这个scale与aspect。我想请问一下，博主如何理解RPN中Softmax是对这些形状与尺度的anchor进行分类的？

潜伏在代码中 Re: 2017-08-09 19:40发表

你说anchor的尺寸与比例在这个过程中并没有用到？那我反问一句，RPN层是如何训练的。

FariverHome Re: 2017-08-10 10:27发表

回复潜伏在代码中：这个问题我相明白了，其实就是训练过程rpn_data中的AnchorTagetLayer会生成每个位置的9个anchor来与RPN的预测anchor计算损失。所有在测试阶段，RPN自然生的就是相对于九个位置的anchor的预测bbox了。那我关于第一个问题： 在im_info上进行ROIpooling会得到每个ROI上的特征图。请问一下，这些特征图是依次传入全连接层进行计算，还是一次性全部传入，得到全部ROI的分类回归结果？我感觉答案应该是：第一个ROI部分进行之后全连接层部分的预测，第二个ROI部分再进行。。。以此类推。但是我看了好久的源码都没找到证据，这请问博主怎么理解这个问题的？

liangxiao06 Re: 2017-08-23 16:24发表

回复FariverHome：为什么要分别传进去，测试的时候生成的RPN窗口是固定的300个，参数是固定一样多的，肯定一起进全连接层

yyrsophie 11楼 2017-07-25 17:44发表

感谢博主，这是我看的写的最明白的关于faster的博客了

eni2396 10楼 2017-07-19 21:39发表

博主你好，我想请教faster-rcnn的检测速率的问题。我用的数据是自己标注的，图片分辨率1920*1080，只检测一类物体，一张图片中可能有多个目标。在Windows在matlab版本的检测速度是20fps；同样的硬件，换成Ubuntu下python版的速度就变成5fps。为什么速度差别会这么大呢？如果换成CPU（至强E5），15s才检测完一张图片。现在需要把模型移植到无人机上，无人机上的开发板rk3399无法用GPU，该开发板CPU的检测速度是30秒/张，即使降低图片的分辨率，速度依然是30秒/帧。有什么方法能实现在CPU下的加速，实现实时检测呢？

DoveJay 9楼 2017-07-19 16:18发表

博主，准确的说，应该是中科大的任少卿在2015年提出了Faster-Rcnn，Ross G是第三作者

whyguu 8楼 2017-07-13 23:15发表

赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞赞，一百个赞。

琴晏 7楼 2017-07-10 11:52发表

感谢楼主的很赞博客。但是我有些问题：
1)在章节2.4中，通过线性回归获得dx(A)，dy(A)，dw(A)，dh(A)，输入为什么是feature map呢？不应该是anchor?
2)续上一问题，如果一个像素点是foreground，那么他的9个anchors都需要回归到GT吗？还是说只要其中的一个anchor回归到GT就可以？

xvshiping1 6楼 2017-06-30 17:33发表

谢谢楼主分享，看了一下午楼主的博客，有个问题想咨询下。conv5出来特征图(m*n)*256，RPN通过一个3*3的滑动窗口做卷积生成一个

关闭

(m*n)*256。我想问的是不是通过256个（3*3）*256卷积核将conv5输出的的特征图做卷积，然后生成256个m*n特征图？。

潜伏在代码中　Re: 2017-06-30 20:04发表

是的。其实就是到conv5输出(m*n)*p大小的特征图，对于zf model：p=256，对于VGG：p=512。然后pad=1，变为((m+2)*(n+2))*p，再经过256个3x3卷积，生成了256个m*n特征图。

xvshiping1　Re: 2017-07-04 16:48发表

回复潜伏在代码中：谢谢博主的回答

lalalazds　5楼 2017-06-28 19:00发表

写的真好，明白了很多。

Q唐Q　4楼 2017-06-27 17:22发表

首先，万分感谢博主的分享，这是我觉得faster rcnn讲得最好最详细的一篇博客。
我想问一下，为什么我用画图工具画出的网络结构为什么和博主画的不一样，有些地方的初入很大。
我是将py-faster-rcnn\models\pascal_voc\VGG16\faster_rcnn_alt_opt目录下的stage1_rpn_train.pt、rpn_test.pt、stage1_fast_rcnn_train.pt三个文件内容输入到画图工具中。

DOUBI2012　3楼 2017-06-13 10:25发表

博主，请问一下：
”foreground softmax scores由大到小排序fg anchors，提取前post_nms_topN(e.g. 300)结果作为proposal输出“
1、这里是按照之前在RPN中的重叠区域IOU形成score进行排序的吗？
2、那在RPN中选取的负样本有什么用，参与训练吗？

潜伏在代码中　Re: 2017-06-13 15:20发表

1. RPN输出的proposal按照softmax scores排序，显然score越高越可能包含目标
2. RPN中的softmax要分类foreground和background，所以需要用正+负样本监督训练softmax，这在loss中体现的很明显

ddhdzt　2楼 2017-05-11 18:29发表

博主的博客写的很赞。我一直不理解为什么要用3*3的窗口大小，感觉窗口大小没有关系，因为定义archor的时候大小是固定的；然后就是1*1的卷积核在RPN网络中的分类层和回归层有什么作用

求是07　Re: 2017-07-19 20:27发表

回复ddhdzt：和我的图标一样哈哈

qq_38269747　1楼 2017-04-11 13:40发表

感谢楼主的博客。但是我有些问题：
1)假定经过卷积层后生成的特征图大小为m*n*256，生成的anchor数量也应该是m*n*9吗？
2)我看论文里说训练里只提取了256个正负anchor进行训练，那么是不是说生成的anchor、进行了筛选？如果是这样，那测试的时候怎么处理？

潜伏在代码中　Re: 2017-04-11 14:22发表

关闭

1.第一个问题，论文中有原话：
By default we use 3 scales and 3 aspect ratios, yielding k = 9 anchors at each sliding position. For a convolutional feature map of a size W × H (typically ~2,400), there are W Hk anchors in total.
2. 第二个问题，代码中有注释：
sort all (proposal, score) pairs by score from highest to lowest，所有的anchors都过网络，然后取score高的就行了啊
（不过你提醒我，图7的解释有误，已经修改了）

qq_38269747 Re: 2017-04-11 15:19发表

回复潜伏在代码中：那么对于anchor的生成，conv5层仍有256张特征图，而从作者的图上(对应博主的图7)来看，生成的结果仍是256张特征图？我觉得直接从conv5的一张特征图提取出候选框坐标不就行了吗，这样输出的特征图只要一张，每个点是一个1*(2*9+4*9)=1*54的向量？

潜伏在代码中 Re: 2017-04-11 17:31发表

回复qq_38269747：如果这么做，99%的有效信息被丢弃了，不用想都知道效果很烂。。。