

经过R-CNN和Fast RCNN的积淀，Ross B. Girshick在2016年提出了新的Faster RCNN，在结构上，Faster RCNN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。

目录

1 Conv layers

2 Region Proposal Networks(RPN)

- 2.1 多通道图像卷积基础知识介绍
- 2.2 anchors
- 2.3 softmax判定positive与negative
- 2.4 bounding box regression原理
- 2.5 对proposals进行bounding box regression
- 2.6 Proposal Layer

3 RoI pooling

- 3.1 为何需要RoI Pooling
- 3.2 RoI Pooling原理

4 Classification

5 Faster RCNN训练

- 5.1 训练RPN网络
- 5.2 通过训练好的RPN网络收集proposals
- 5.3 训练Faster RCNN网络

Questions and Answer

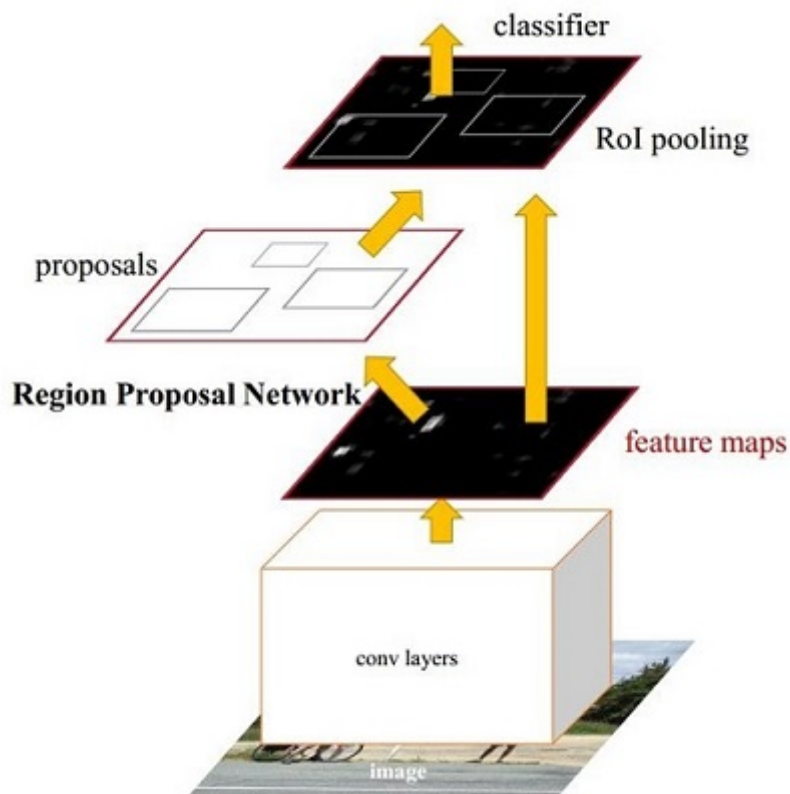


图1 Faster RCNN基本结构 (来自原论文)

依作者看来，如图1，Faster RCNN其实可以分为4个主要内容：

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。

Region Proposal Networks。RPN网络用于生成region proposals。该层通过softmax判断anchors属于positive或者negative，再利用bounding box regression修正anchors获得精确的proposals。

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。

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 R-CNN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positive anchors和对应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）。

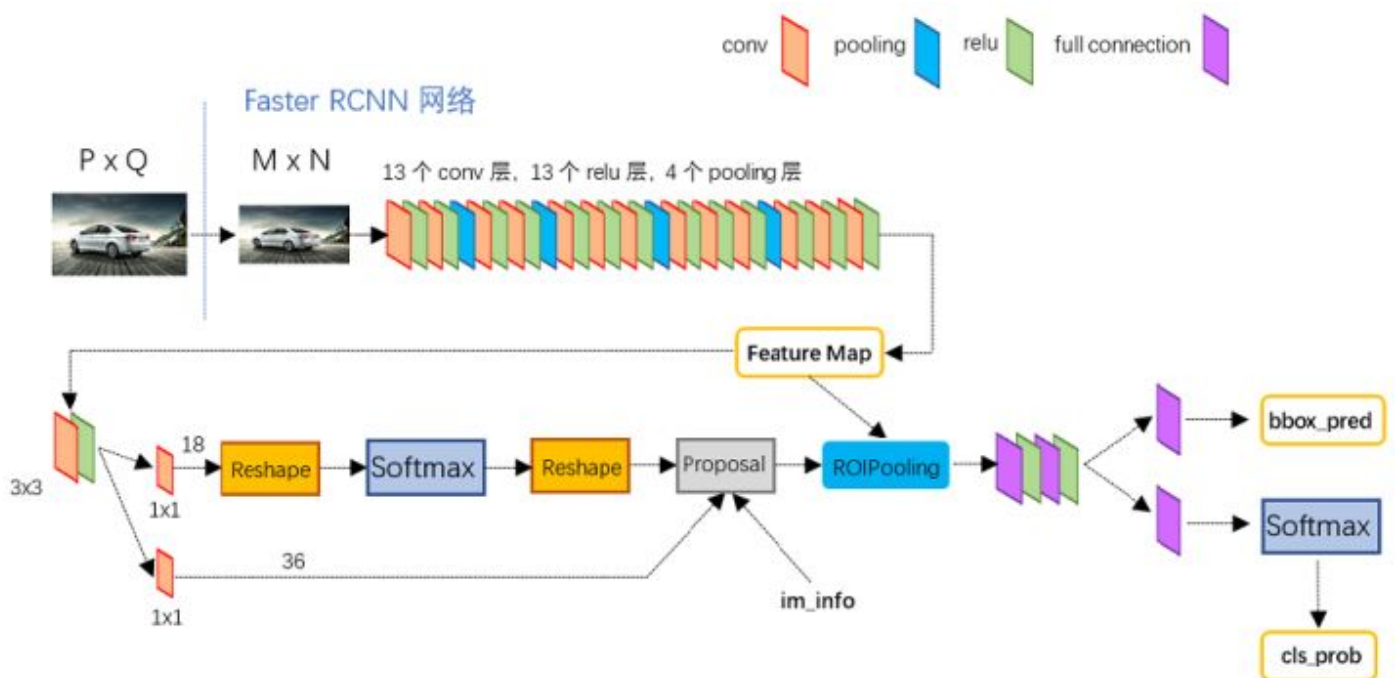
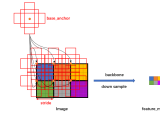


图2 faster_rcnn_test.pt网络结构（pascal_voc/VGG16/faster_rcnn_alt_opt/faster_rcnn_test.pt）

本文不会讨论任何关于R-CNN家族的历史，分析清楚最新的Faster R-CNN就够了，并不需要追溯到那么久。实话说我也不了解R-CNN，更不关心。有空不如看看新算法。

新出炉的pytorch官方Faster RCNN代码导读：



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：

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

所有的pooling层都是：kernel_size=2, pad=0, 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 $(M+2) \times (N+2)$ 大小，再做3x3卷积后输出 $M \times N$ 。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：

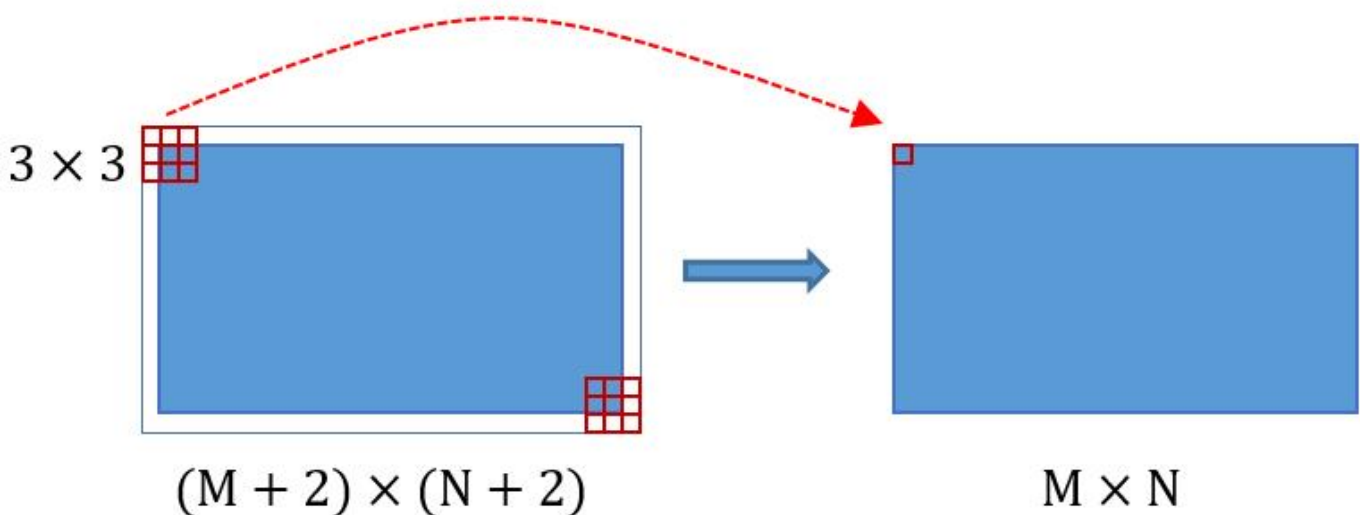


图3 卷积示意图

类似的是，Conv layers中的pooling层kernel_size=2, stride=2。这样每个经过pooling层的 $M \times N$ 矩阵，都会变为 $(M/2) \times (N/2)$ 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 $M \times N$ 大小的矩阵经过Conv layers固定变为 $(M/16) \times (N/16)$ ！这样Conv layers生成的feature map中都可以和原图对应起来。

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R-CNN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 R-CNN的巨大优势，能极大提升检测框的生成速度。

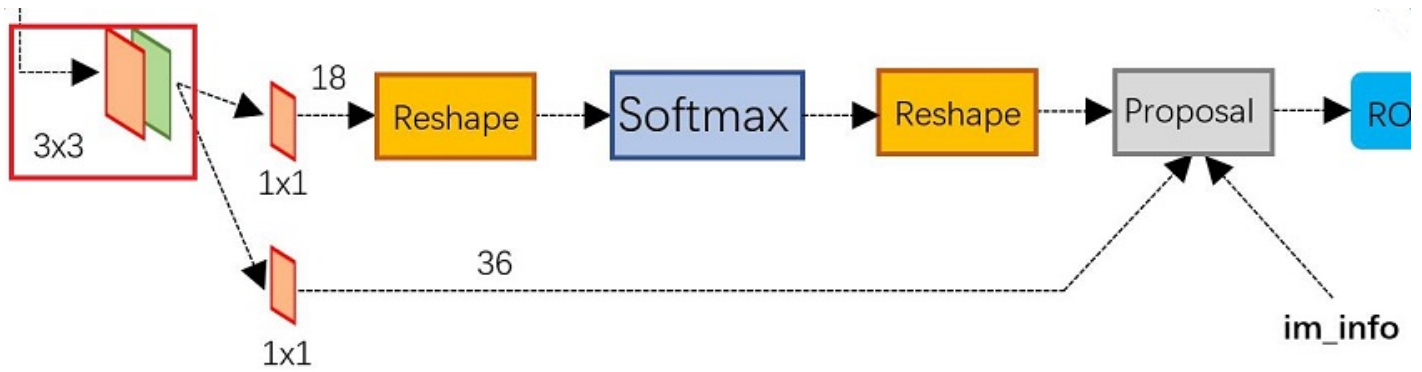


图4 RPN网络结构

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positive和negative分类，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positive anchors和对应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。

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；
对于多通道图像+多卷积核做卷积，计算方式如下：

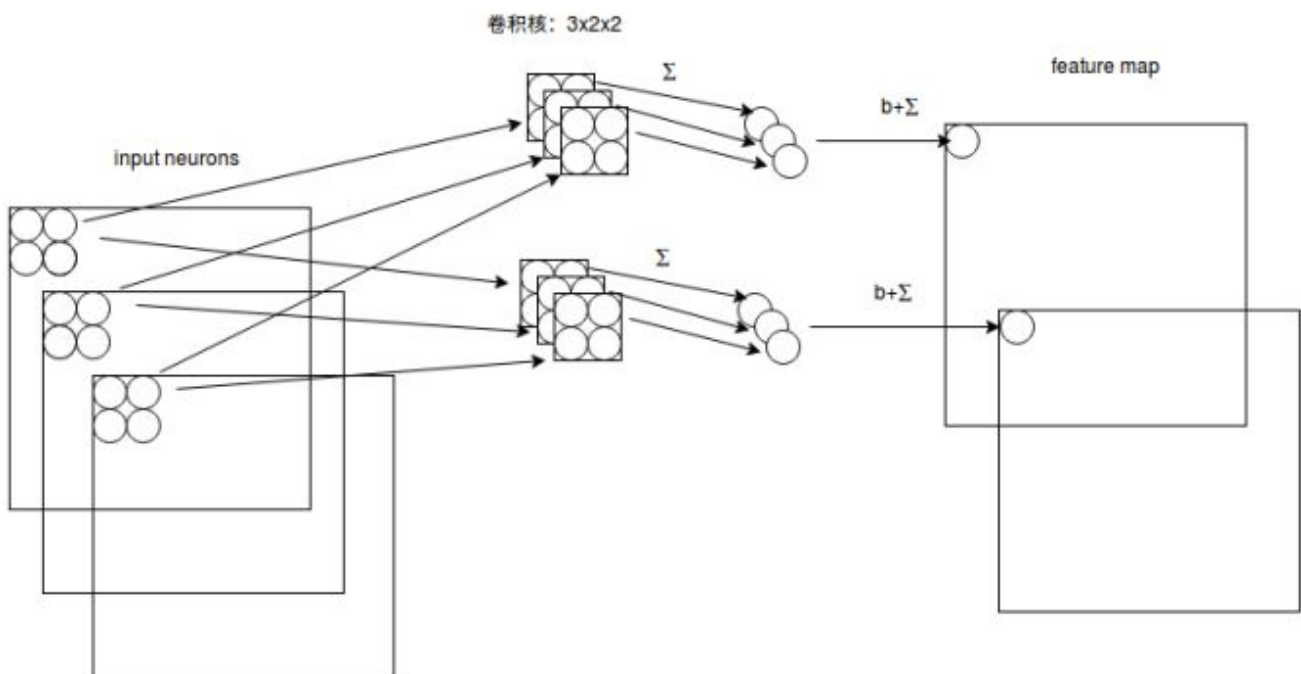


图5 多通道卷积计算方式

如图5，输入有3个通道，同时有2个卷积核。对于每个卷积核，先在输入3个通道分别作卷积，再将3个通道结果加起来得到卷积输出。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：

```
[[ -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 (x_1, y_1, x_2, y_2) 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 $\text{width:height} \in \{1:1, 1:2, 2:1\}$ 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。

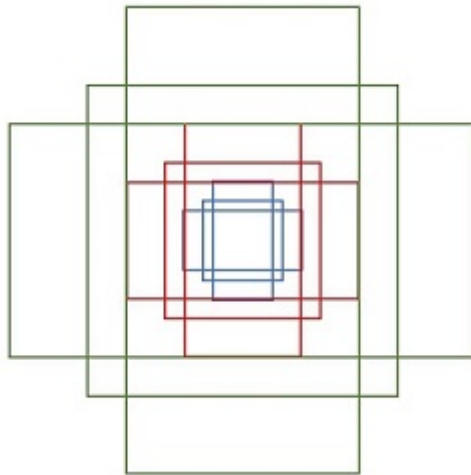


图6 anchors示意图

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。

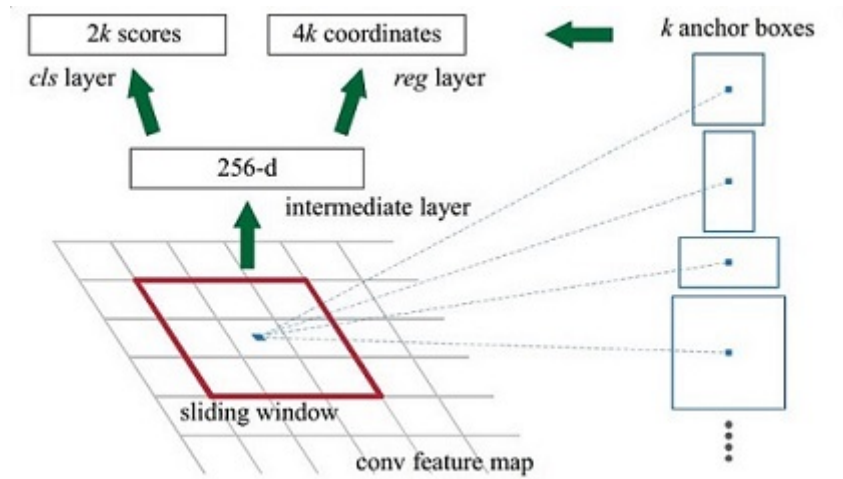


图7

解释一下上面这张图的数字。

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-dimensions

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）

假设在conv5 feature map中每个点上有k个anchor（默认k=9），而每个anchor要分positive和negative，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

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positive 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。

其实RPN最终就是在原图尺度上，设置了密密麻麻的候选Anchor。然后用cnn去判断哪些Anchor是里面有目标的positive anchor，哪些是没目标的negative anchor。所以，仅仅是个二分类而已！

那么Anchor一共有多少个？原图800x600，VGG下采样16倍，feature map每个点设置9个Anchor，所以：

$$\text{ceil}(800/16) \times \text{ceil}(600/16) \times 9 = 50 \times 38 \times 9 = 17100 \quad (1)$$

其中ceil()表示向上取整，是因为VGG输出的feature map size= 50*38。

Generate Anchors

Given:

- Set of aspect ratios (0.5, 1, 2)
- Stride length (downscaling performed by resnet head: 16)
- Anchor Scales (8, 16, 32)

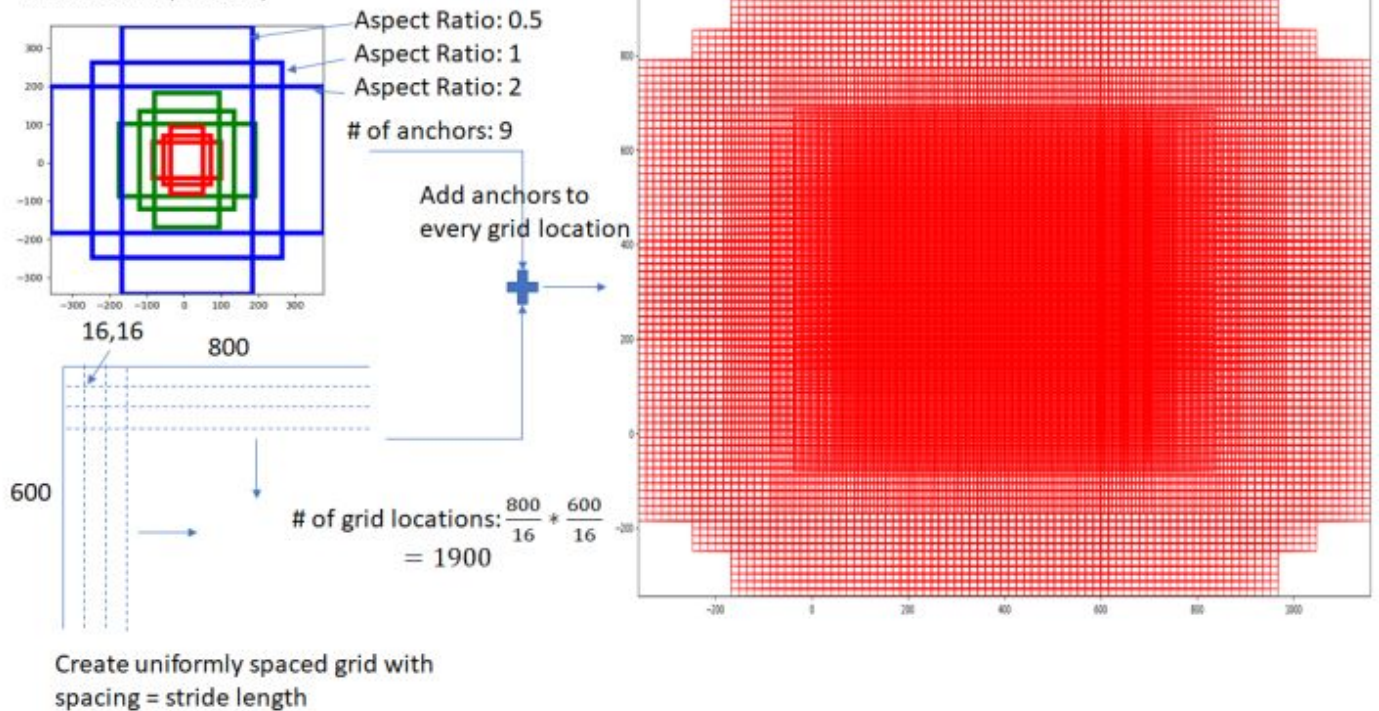


图8 Generate Anchors

2.3 softmax判定positive与negative

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9：

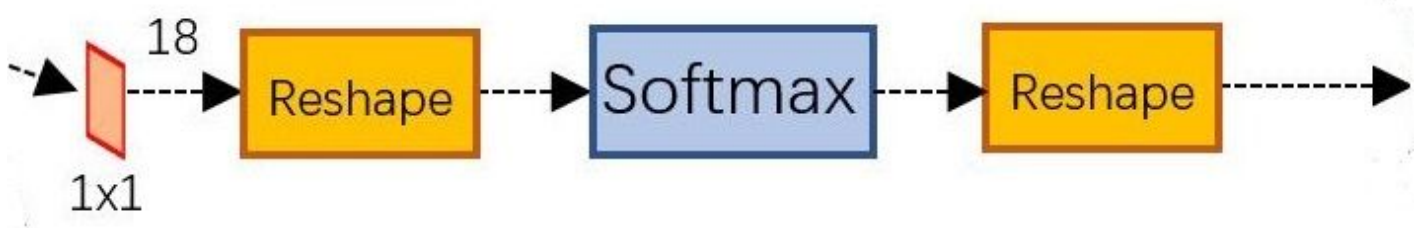


图9 RPN中判定positive/negative网络结构

该1x1卷积的caffe prototxt定义如下：

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

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positive和negative，所有这些信息都保存WxHx(9*2)大小的矩阵。为何这样做？后面接softmax分类获得positive 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positive 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positive/negative anchors的矩阵，其在caffe blob中的存储形式为[1, 2x9, H, W]。而在softmax分类时需要进行positive/negative二分类，所以reshape layer会将其变为[1, 2, 9xH, 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：

```
"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positive anchors作为候选区域（另外也有实现用sigmoid代替softmax，输出[1, 1, 9xH, W]后接sigmoid进行positive/negative二分类，原理一样）。

2.4 bounding box regression原理

如图9所示绿色框为飞机的Ground Truth(GT)，红色为提取的positive anchors，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positive anchors和GT更加接近。

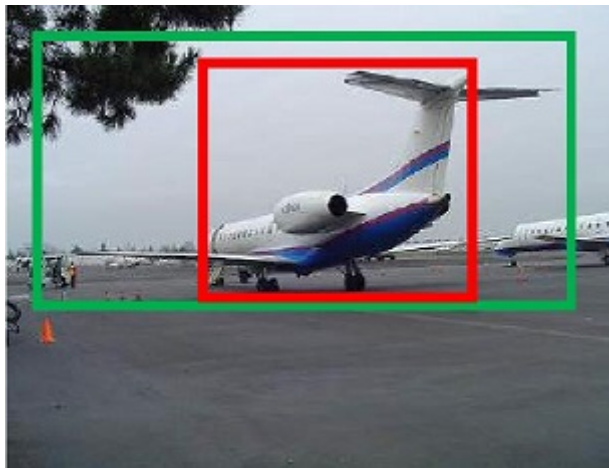


图10

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 11，红色的框A代表原始的positive 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 $A = (A_x, A_y, A_w, A_h)$ 和 $GT = [G_x, G_y, G_w, G_h]$

寻找一种变换F, 使得: $F(A_x, A_y, A_w, A_h) = (G'_x, G'_y, G'_w, G'_h)$, 其中 $(G'_x, G'_y, G'_w, G'_h) \approx (G_x, G_y, G_w, G_h)$

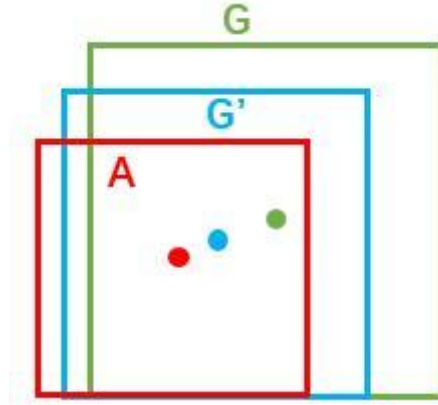


图11

那么经过何种变换F才能从图10中的anchor A变为G'呢? 比较简单的思路就是:

先做平移

$$G'_x = A_w \cdot d_x(A) + A_x \quad (2)$$

$$G'_y = A_h \cdot d_y(A) + A_y \quad (3)$$

再做缩放

$$G'_w = A_w \cdot \exp(d_w(A)) \quad (4)$$

$$G'_h = A_h \cdot \exp(d_h(A)) \quad (5)$$

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)。

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cnn feature map, 定义为 Φ ; 同时还有训练传入A与GT之间的变换量, 即 (t_x, t_y, t_w, t_h) 。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:

$$d_*(A) = W_*^T \cdot \phi(A) \quad (6)$$

其中 $\phi(A)$ 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 $d_*(A)$ 与真实值 t_* 差距最小, 设计L1损失函数:

$$\text{Loss} = \sum_i^N |t_*^i - W_*^T \cdot \phi(A^i)| \quad (7)$$

函数优化目标为:

$$\hat{W}_* = \operatorname{argmin}_{W_*} \sum_i^n |t_*^i - W_*^T \cdot \phi(A^i)| + \lambda \|W_*\| \quad (8)$$

为了方便描述, 这里以L1损失为例介绍, 而真实情况中一般使用smooth-L1损失。

需要说明, 只有在GT与需要回归框位置比较接近时, 才可近似认为上述线性变换成立。说完原理, 对应于Faster RCNN原文, positive anchor与ground truth之间的平移量 (t_x, t_y) 与尺度因子 (t_w, t_h) 如下:

$$t_x = (x - x_a)/w_a \quad t_y = (y - y_a)/h_a \quad (9)$$

$$t_w = \log(w/w_a) \quad t_h = \log(h/h_a) \quad (10)$$

对于训练bounding box regression网络回归分支, 输入是cnn feature Φ , 监督信号是Anchor与GT的差距 (t_x, t_y, t_w, t_h) , 即训练目标是: 输入 Φ 的情况下使网络输出与监督信号尽可能接近。那么当bounding box regression工作时, 再输入 Φ 时, 回归网络分支的输出就是每个Anchor的平移量和变换尺度 (t_x, t_y, t_w, t_h) , 显然即可用来修正Anchor位置了。

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12。



图12 RPN中的bbox reg

先来看一看上图11中1x1卷积的caffe prototxt定义:

```

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, 4x9, 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

$$[d_x(A), d_y(A), d_w(A), d_h(A)] \quad (11)$$

变换量。

回到图8, VGG输出 $50 * 38 * 512$ 的特征, 对应设置 $50 * 38 * k$ 个anchors, 而RPN输出:

大小为 $50 * 38 * 2k$ 的positive/negative softmax分类特征矩阵

大小为 $50 * 38 * 4k$ 的regression坐标回归特征矩阵

恰好满足RPN完成positive/negative分类+bounding box regression坐标回归。

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positive 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:

```

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: positive vs negative 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P×Q图像，传入Faster RCNN前首先reshape到固定M×N，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W×H=(M/16)×(N/16)大小，其中feature_stride=16则保存了该信息，用于计算anchor偏移量。

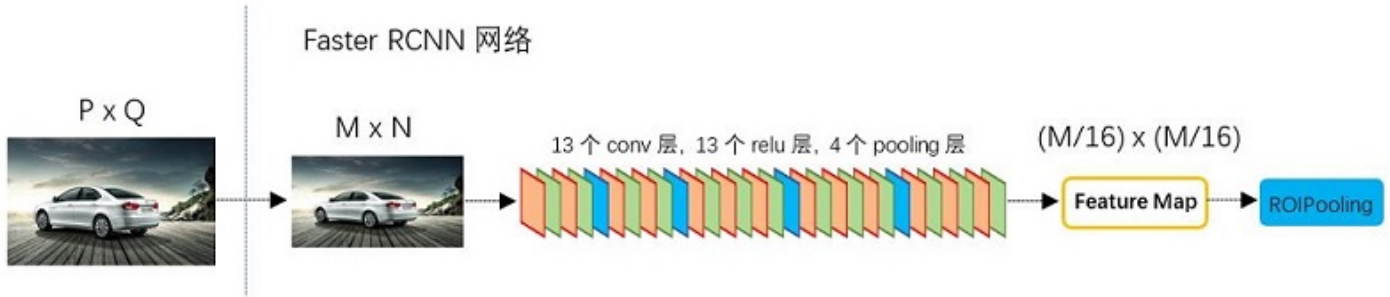


图13

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

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)

按照输入的positive 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positive anchors

限定超出图像边界的positive anchors为图像边界，防止后续roi pooling时proposal超出图像边界 (见文章底部QA部分图21)

剔除尺寸非常小的positive anchors

对剩余的positive anchors进行NMS (nonmaximum suppression)

Proposal Layer有3个输入：positive和negative anchors分类器结果rpn_cls_prob_reshape，对应的bbox reg的(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M×N输入图像尺度的，这点在后续网络中 useful。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positive anchors -> bbox reg回归positive 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：

原始的feature maps

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和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：

从图像中crop一部分传入网络
将图像warp成需要的大小后传入网络



图14 crop与warp破坏图像原有结构信息

两种办法的示意图如图14，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positive anchors进行bounding 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 R-CNN中提出了RoI Pooling解决这个问题。不过RoI Pooling确实是从Spatial Pyramid Pooling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：

```
layer {
  name: "roi_pool5"
  type: "ROI Pooling"
  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和pooled_h，另外一个参数spatial_scale认真阅读的读者肯定已经知道知道用途。RoI Pooling layer forward过程：

由于proposal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 map尺度；
再将每个proposal对应的feature map区域水平分为 $\text{pooled_w} \times \text{pooled_h}$ 的网格；
对网格的每一份都进行max pooling处理。

这样处理后，即使大小不同的proposal输出结果都是 $\text{pooled_w} \times \text{pooled_h}$ 固定大小，实现了固定长度输出。

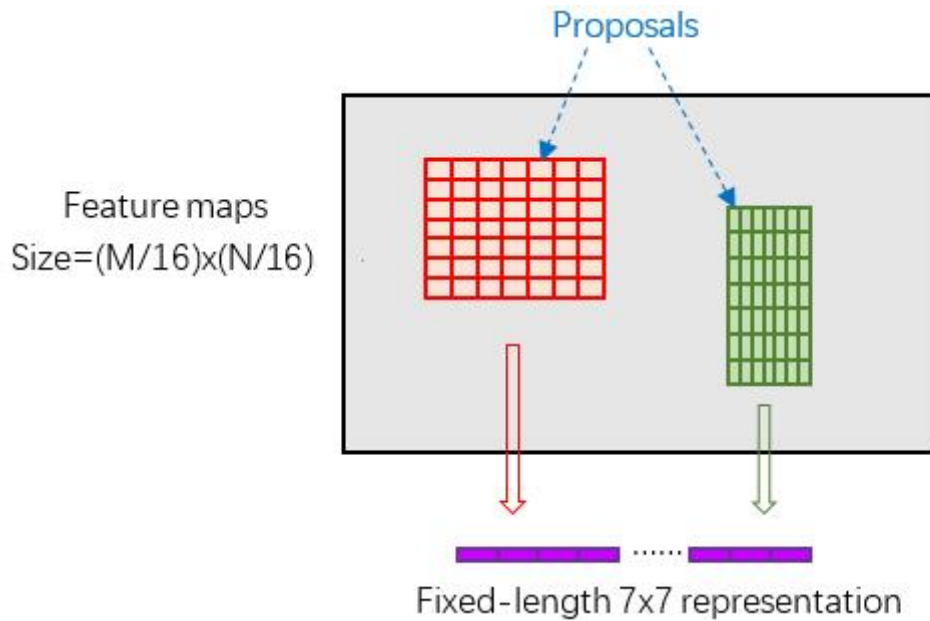


图15 proposal示意图

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16。

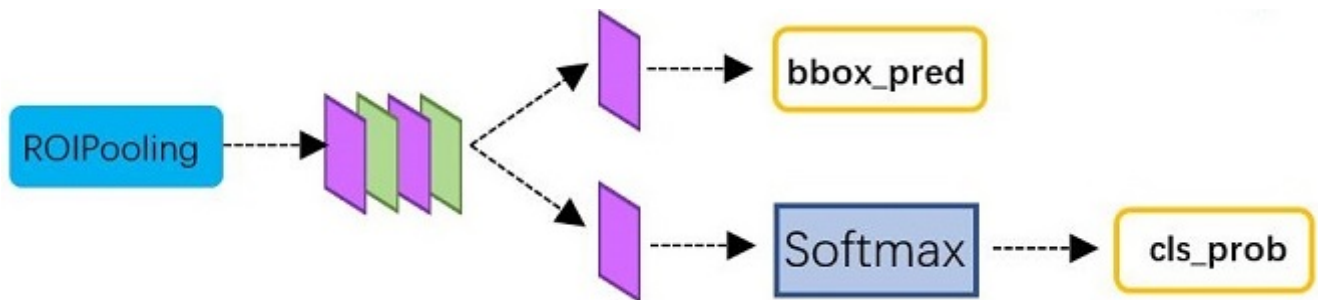


图16 Classification部分网络结构图

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

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
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17，

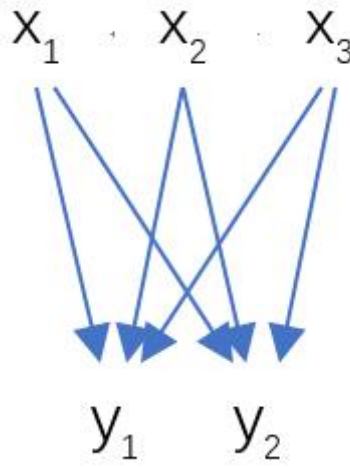


图17 全连接层示意图

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 R-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：

在已经训练好的model上，训练RPN网络，对应stage1_rpn_train.pt

利用步骤1中训练好的RPN网络，收集proposals，对应rpn_test.pt

第一次训练Fast RCNN网络，对应stage1_fast_rcnn_train.pt

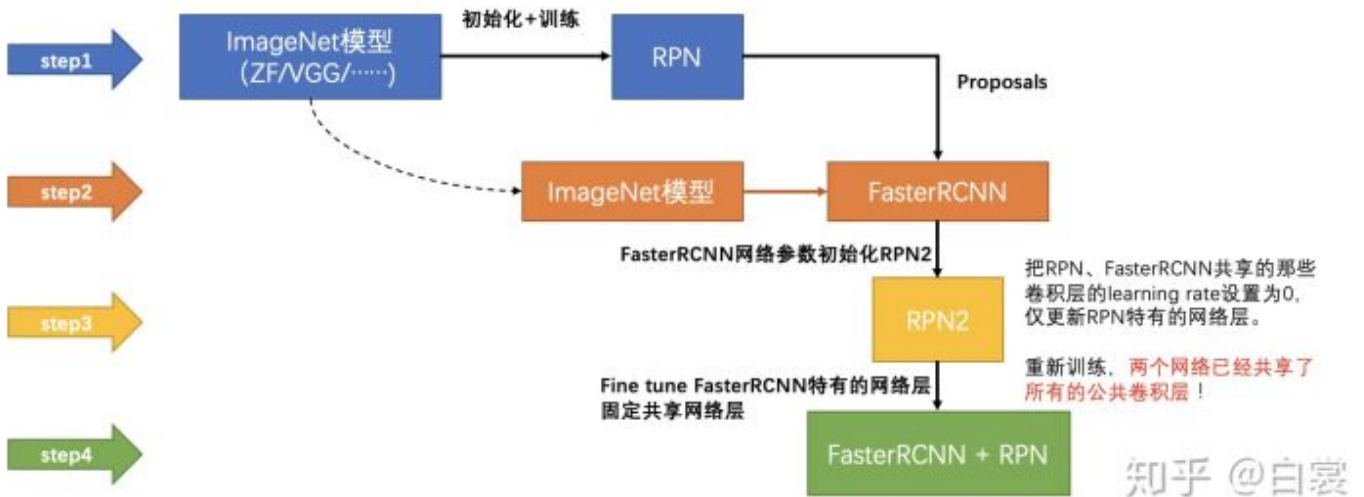
第二次训练RPN网络，对应stage2_rpn_train.pt

再次利用步骤4中训练好的RPN网络，收集proposals，对应rpn_test.pt

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。

下面是一张训练过程流程图，应该更加清晰：



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19。

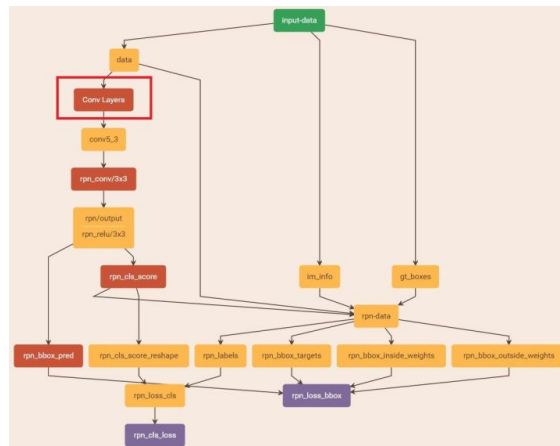


图19 stage1_rpn_train.pt (考虑图片大小, Conv Layers中所有的层都画在一起了, 如红圈所示, 后续图都如此处理)

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:

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(p_i, p_i^*) + \lambda \frac{1}{N_{\text{reg}}} \sum_i p_i^* L_{\text{reg}}(t_i, t_i^*) \quad (12)$$

上述公式中 i 表示anchors index, p_i 表示positive softmax probability, 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positive, $p_i^* = 1$; 反之IoU<0.3时, 认为是该anchor是negative, $p_i^* = 0$; 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positive anchor对应的GT box。可以看到, 整个Loss分为2部分:

cls loss, 即rpn_cls_loss层计算的softmax loss, 用于分类anchors为positive与negative的网络训

练

reg loss, 即rpn_loss_bbox层计算的smooth 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positive anchors的回归（其实在回归中也完全没必要去关心negative）。

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 $N_{\text{cls}} = 256$, $N_{\text{reg}} = 2400$ 时设置 $\lambda = \frac{N_{\text{reg}}}{N_{\text{cls}}} \approx 10$), 使总的网络Loss计算过程中能够均匀考虑2种Loss。这里比较重要的是 L_{reg} 使用的smooth L1 loss, 计算公式如下:

$$L_{\text{reg}}(t_i, t_i^*) = \sum_{i \in \{x, y, w, h\}} \text{smooth}_{\text{L1}}(t_i - t_i^*) \quad (13)$$

$$\text{smooth}_{\text{L1}}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \quad (14)$$

了解数学原理后, 反过来看图18:

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

对于rpn_loss_cls, 输入的rpn_cls_scores_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

对于rpn_loss_bbox, 输入的rpn_bbox_pred和rpn_bbox_targets分别对应 t 与 t^* , rpn_bbox_inside_weights对应 p^* , rpn_bbox_outside_weights未用到 (从smooth_L1_Loss layer代码中可以看到), 而 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positive softmax probability, 如图20, 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。

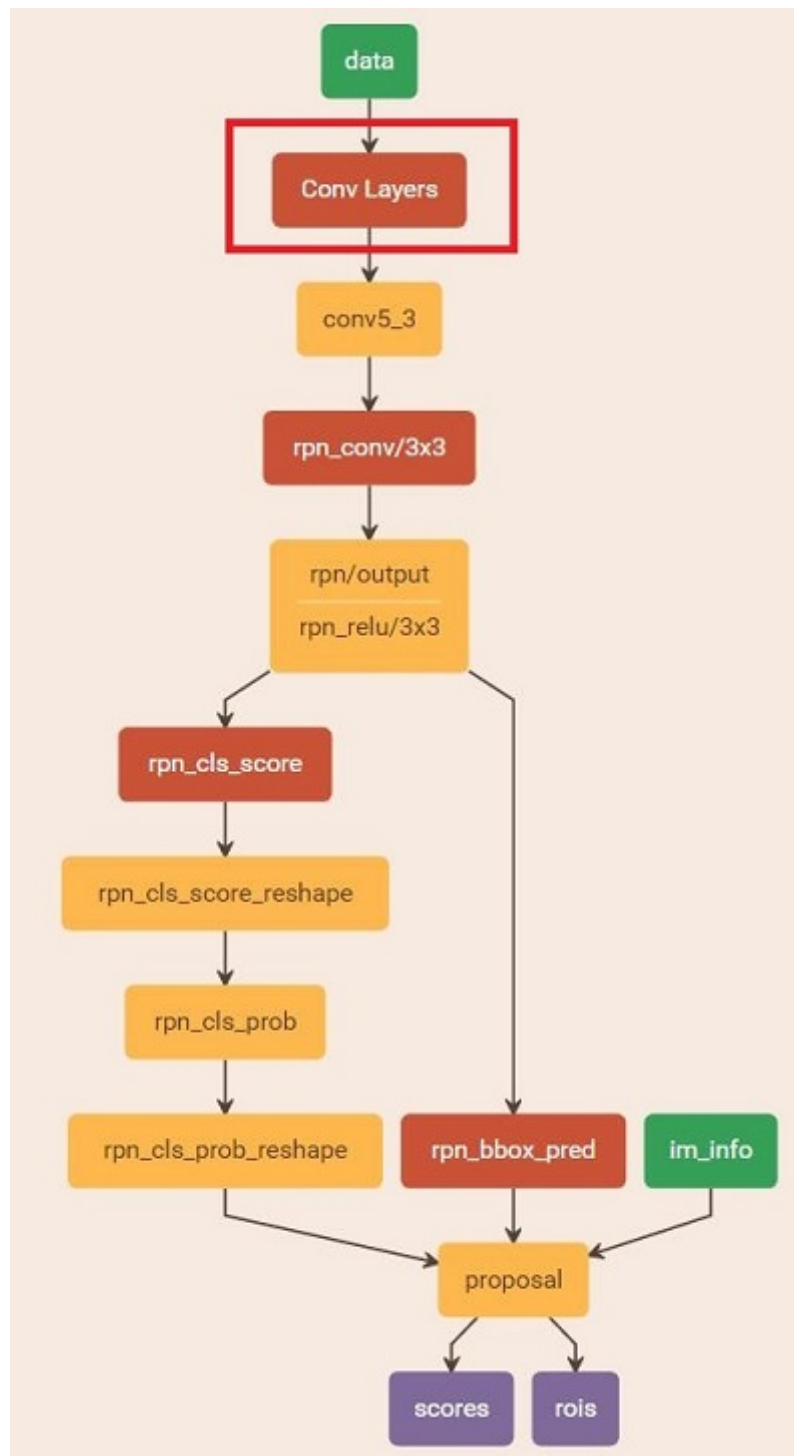


图20 rpn_test.pt

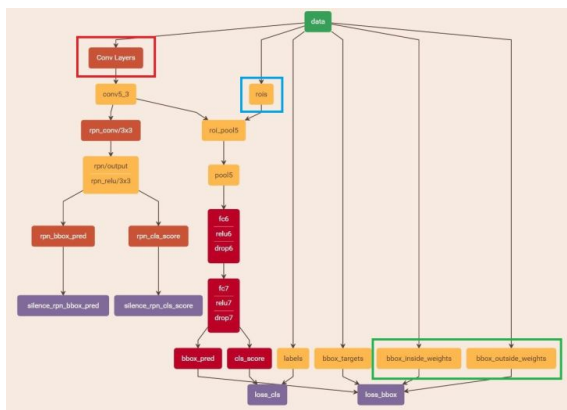
5.3 训练Faster RCNN网络

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

将提取的proposals作为rois传入网络，如图21蓝框

计算bbox_inside_weights+bbox_outside_weights，作用与RPN一样，传入smooth_L1_loss layer，如图21绿框

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



之后的stage2训练都是大同小异，不再赘述了。Faster R-CNN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。

rbgirshick py-faster-rcnn
[github.com](https://github.com/rbgirshick/py-faster-rcnn)



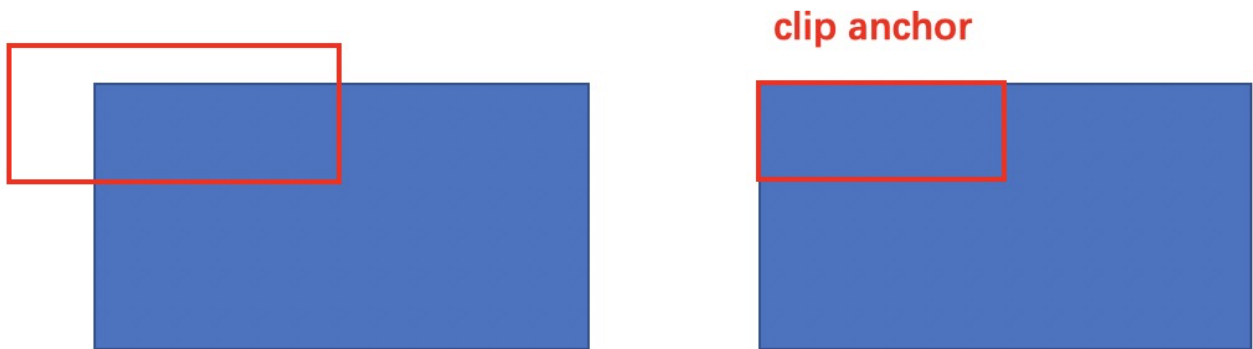
QA

此篇文章初次成文于2016年内部学习分享，再后来经多次修正和完善成为现在的样子。感谢大家一直以来的支持，现在总结常见疑问回答如下：

为什么Anchor坐标中有负数

回顾anchor生成步骤：首先生成9个base anchor，然后通过坐标偏移在 $50 * 38$ 大小的 $\frac{1}{16}$ 下采样FeatureMap每个点都放上这9个base anchor，就形成了 $50 * 38 * k$ 个anchors。至于这9个base anchor坐标是什么其实并不重要，不同代码实现也许不同。

显然这里面有一部分边缘anchors会超出图像边界，而真实中不会有超出图像的目标，所以会有clip anchor步骤。



Anchor到底与网络输出如何对应

VGG输出 $50 * 38 * 512$ 的特征，对应设置 $50 * 38 * k$ 个anchors，而RPN输出 $50 * 38 * 2k$ 的分类特征矩阵和 $50 * 38 * 4k$ 的坐标回归特征矩阵。

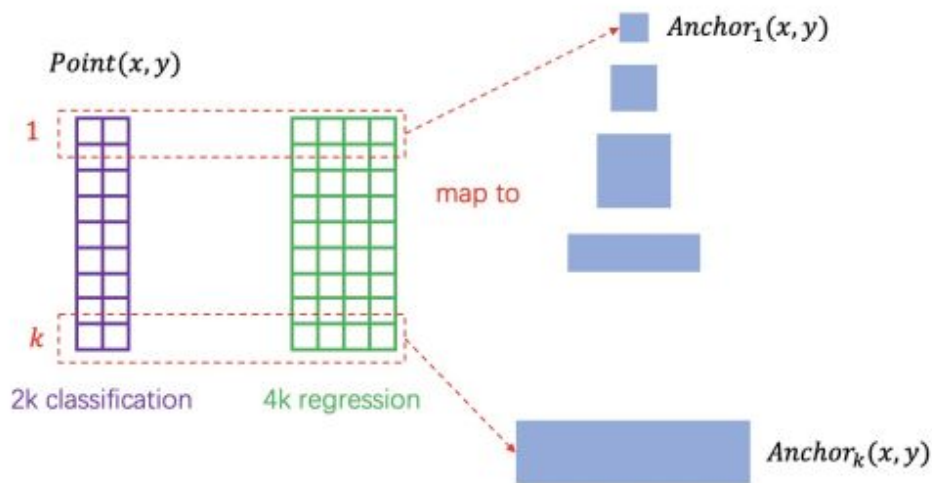


图22 anchor与网络输出如何对应方式

其实在实现过程中，每个点的 $2k$ 个分类特征与 $4k$ 回归特征，与 k 个anchor逐个对应即可，这实际是一种“人为设置的逻辑映射”。当然，也可以不这样设置，但是无论如何都需要保证在训练和测试过程中映射方式必须一致。

为何有ROI Pooling还要把输入图片resize到固定大小的MxN

由于引入ROI Pooling，从原理上说Faster R-CNN确实能够检测任意大小的图片。但是由于在训练的时候需要使用大batch训练网络，而不同大小输入拼batch在实现的时候代码较为复杂，而且当时以Caffe为代表的第一代深度学习框架也不如Tensorflow和PyTorch灵活，所以作者选择了把输入图片resize到固定大小的800x600。这应该算是历史遗留问题。

另外很多问题，都是属于具体实现问题，真诚的建议读者阅读代码自行理解。