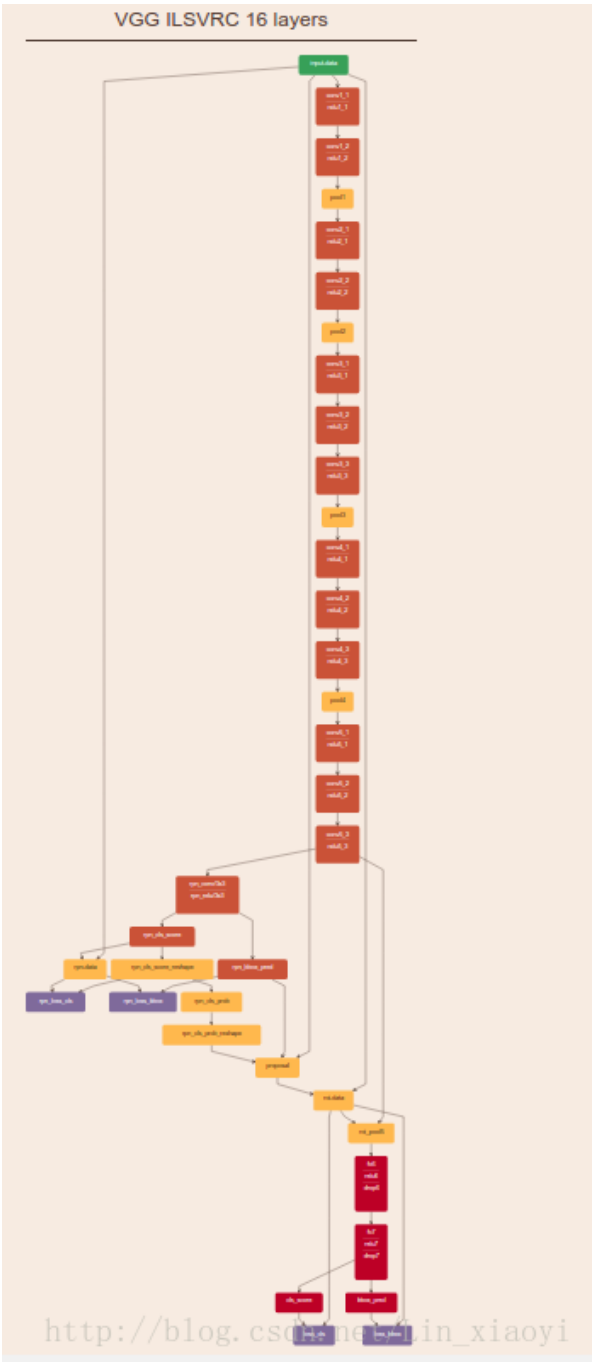
结题报告

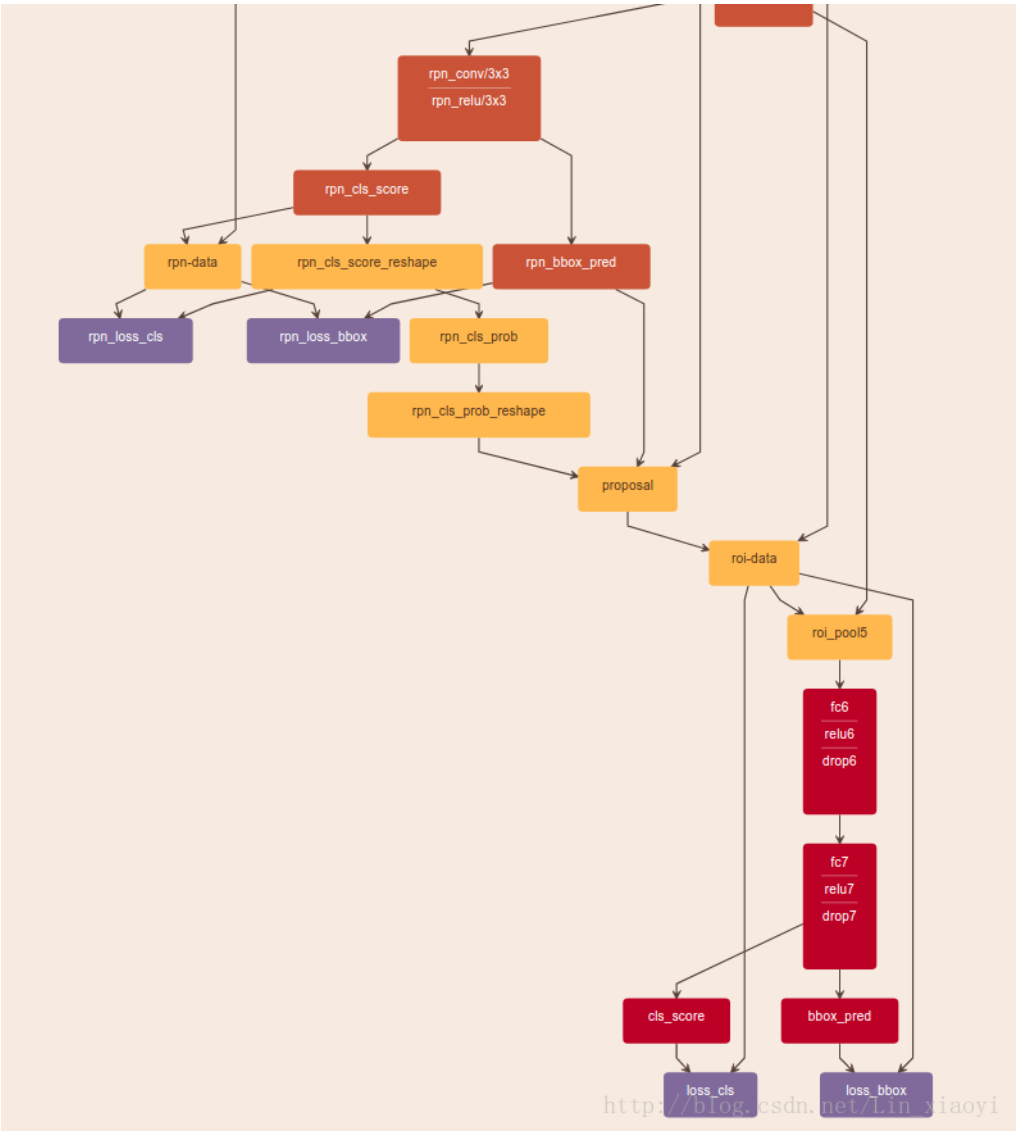
我们采用了 faster-rcnn 算法。

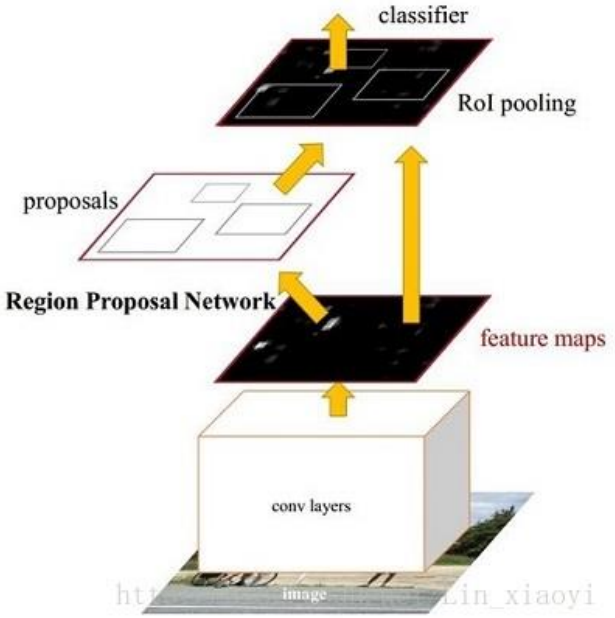
faster-rcnn 原理

论文的 demo 用了 ZF 和 VGG16 的网络结构，本文默认用 VGG16
它的结构图：

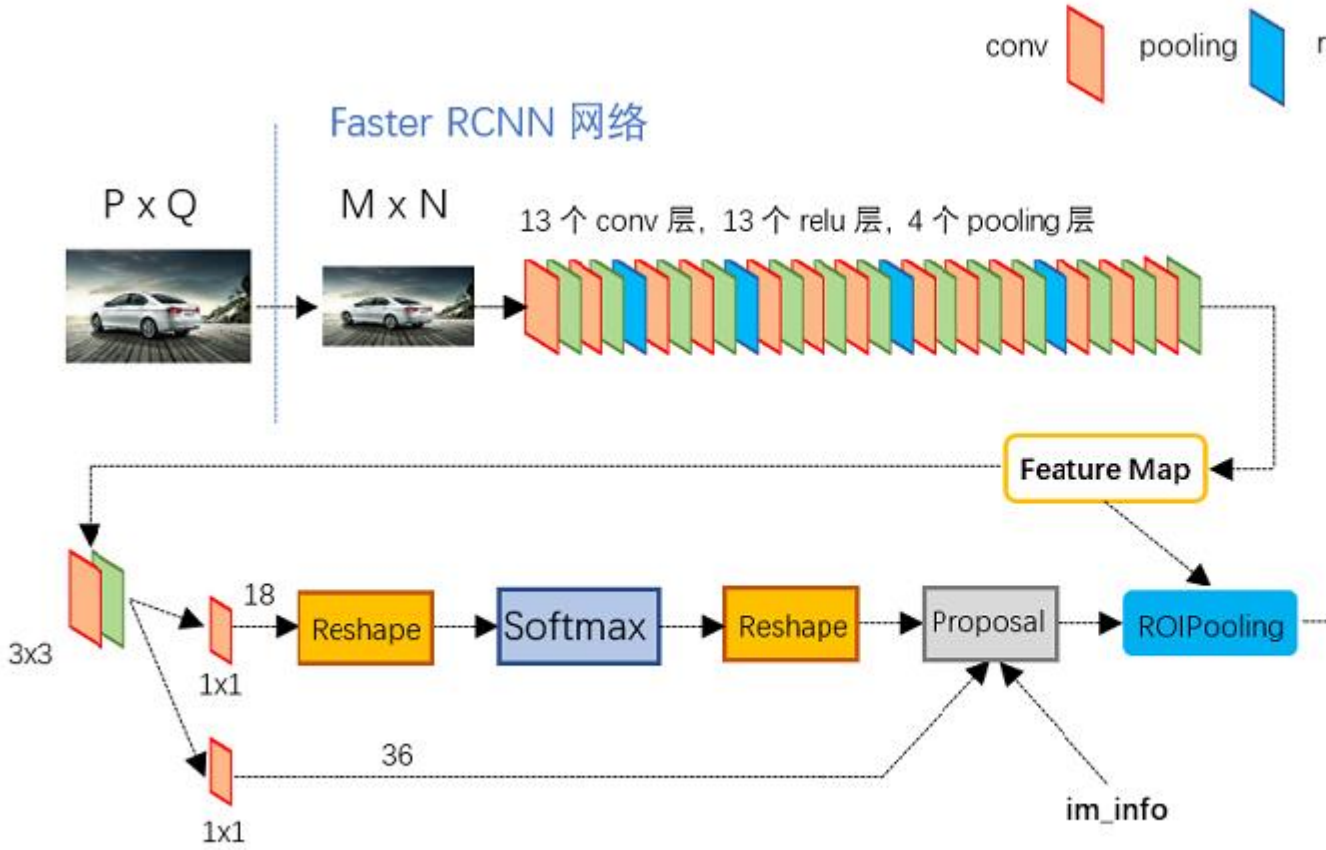


它的前 13 层是用了 VGG 提取特征，主要算法实习是后面几层


rpn_conv/3x3
rpn_relu/3x3
rpn_cls_score
rpn-data
rpn_cls_score_reshape
rpn_bbox_pred
rpn_loss_cls
rpn_loss_bbox
rpn_cls_prob
rpn_cls_prob_reshape
proposal
roi-data
roi_pool5
fc6
relu6
drop6
fc7
relu7
drop7
cls_score
bbox_pred
loss_cls
loss_bbox


classifier
RoI pooling
proposals
Region Proposal Network
feature maps
conv layers
image

test 网络



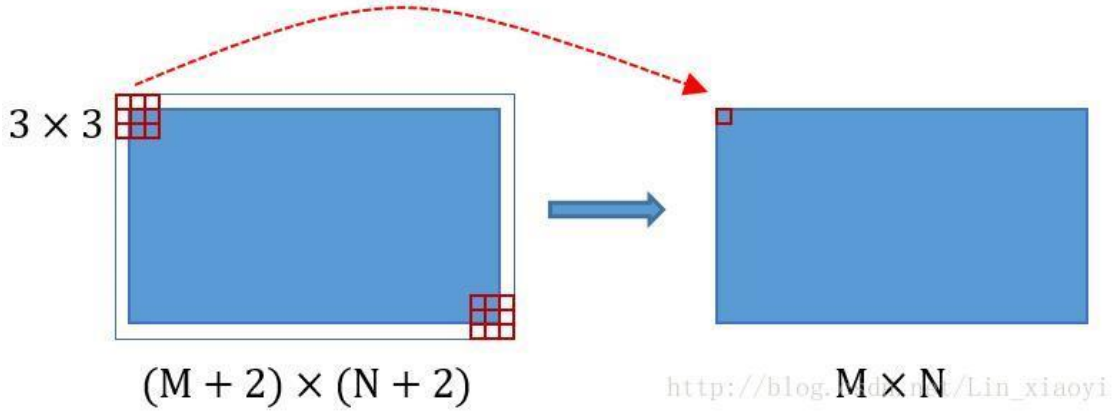
算法步骤

- １．Conv layers.作为一种 cnn 网络目标检测的方法，faster_rcnn 首先使用一组基础 conv+relu+pooling 层提取 image 的 feture map。该 feature map 被共享用于后续的 RPN 层和全连接层。
- ２．Region Proposal Networks.RPN 层是 faster-rcnn 最大的亮点，RPN 网络用于生成 region proposcals.该层通过 softmax 判断 anchors 属于 foreground 或者 background，再利用 box regression 修正 anchors 获得精确的 propocals（anchors 也是作者自己提出来的，后面我们会认真讲）
- ３．Roi Pooling.该层收集输入的 feature map 和 proposcal，综合这些信息提取 proposal feature map，送入后续的全连接层判定目标类别。
- ４． Classification。利用 proposal feature map 计算 proposcal 类别，同时再次 bounding box regression 获得检验框的最终精确地位置

详解

1．Conv layer

在 input-data 层时，作者把原图都 reshape 成 M×N 大小的图片
conv layer 中包含了 conv relu pooling 三种层,就 VGG16 而言，就有 13 个 conv 层，13 个 relu 层， 4 个 pooling 层。在 conv layer 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

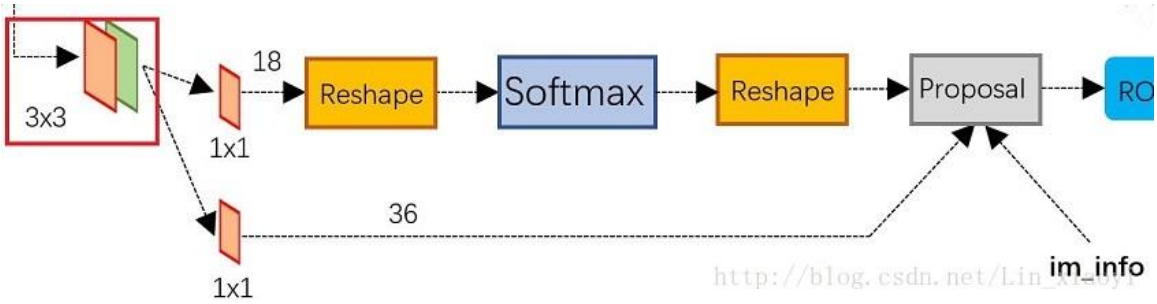


Conv layer 中的 pooling 层 kernel_size=2,stride=2，这样使得经过 pooling 层中 M×N 矩阵，都会变为(M/2)*(N/2)大小。综上所述，在整个 Conv layers 中，conv 和 relu 层不改变输入输出大小，只有 pooling 层使输出长宽都变为输入的 1/2。
那么，一个 MxN 大小的矩阵经过 Conv layers 固定变为(M/16)x(N/16)！这样 Conv layers 生成的 featuure map 中都可以和原图对应起来。

2．Region Propocal 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。

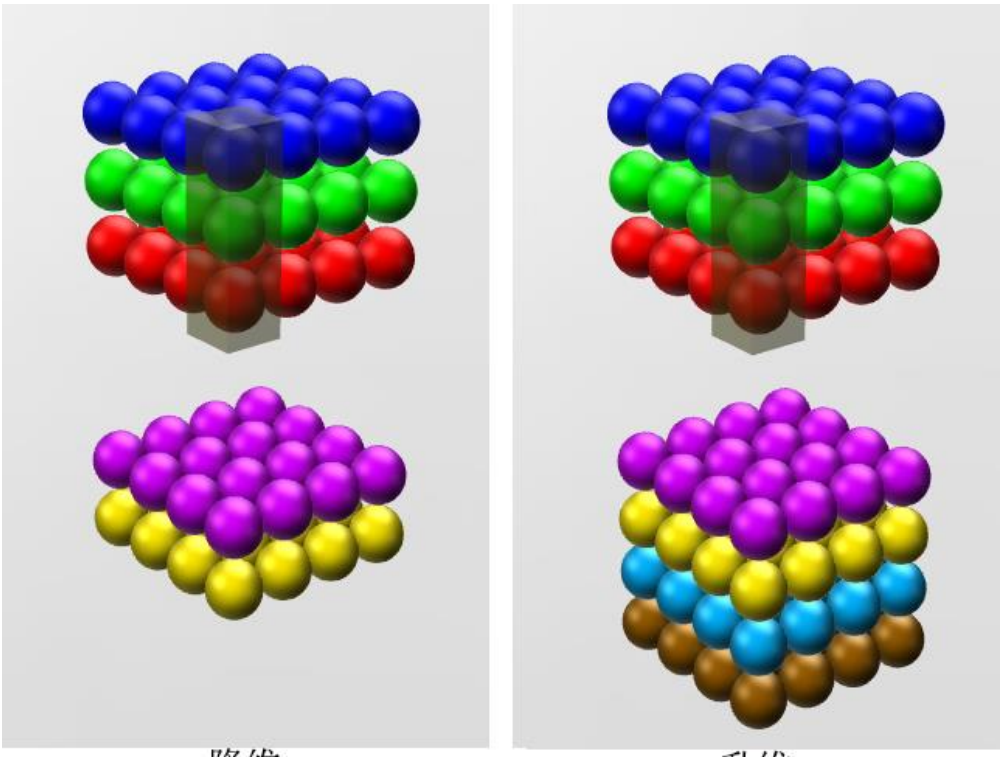


上图中展示了 RPN 网络的具体结构，可以看到，feature map 经过一个 3×3 卷积核卷积后分成了两条线，上面一条通过 softmax 对 anchors 分类获得 foreground 和 background（检测目标是 foregrounnd），因为是 2 分类，所以它的维度是 2k scores。下面那条线是用于计算 anchors 的 bounding box regression 的偏移量，以获得精确的 proposal。它的维度是 4k coordinates。而最后的 proposcal 层则负责综合 foreground anchors 和 bounding box regression 偏移量获取 proposal，同时剔除太小和超出边界的 propocals,其实网络到这个 Proposal Layer 这里，就完成了目标定位的功能

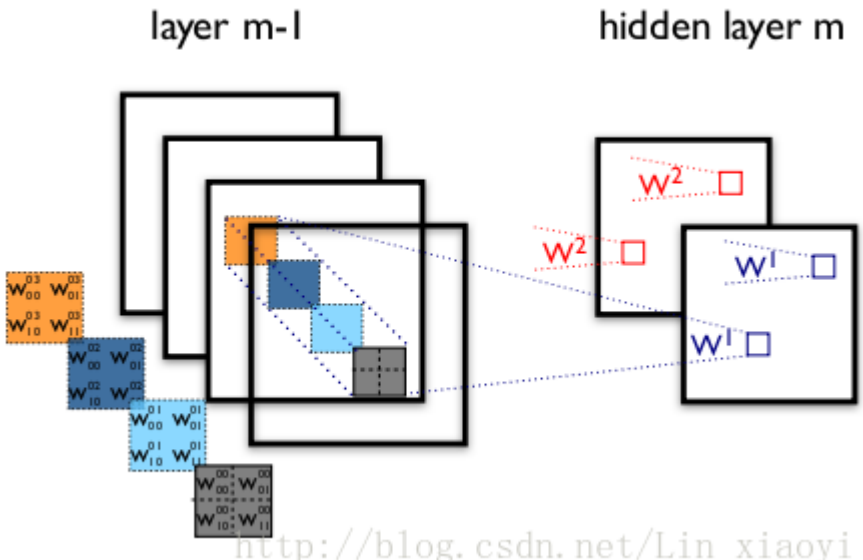
1x1 卷积核,3x3 卷积核

如果卷积的输出输入都只是一个平面，那么 1x1 卷积核并没有什么意义，它是完全不考虑像素与周边其他像素关系。 但卷积的输出输入是长方体，所以 1x1 卷积实际上是对每个像素点，在不同的 channels 上进行线性组合（信息整合），且保留了图片的原有平面结构，调控 depth，从而完成升维或降维的功

能。



- 对于单通道图像+单卷积核做卷积，前面我已经讲过了。
- 对于多通道图像 + 多卷积核做卷积，计算方式如下：



如图，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！

对于多通道图像做 1× 1 卷积，其实就是将输入的图像的每个通道乘以卷积系数加在一起，即相当于把原图像中本来各个独立通道”联通”在一起。

anchors

提到 PRN 网络，就不能不说 anchors。所谓的 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.]

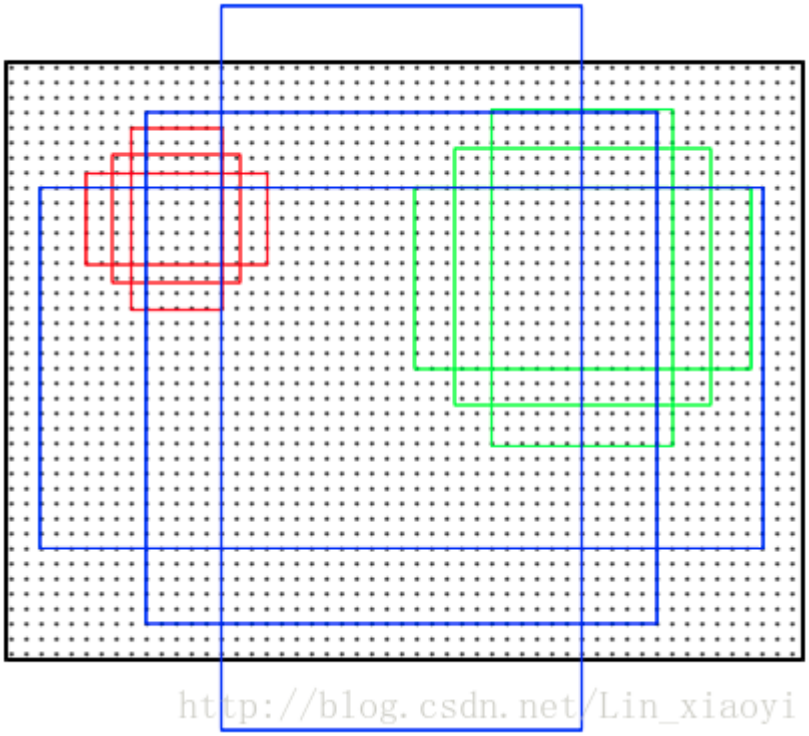
[-36. -80. 51. 95.]

[-80. -168. 95. 183.]

[-168. -344. 183. 359.]]

其中每行的 4 个值（x1,y1,x2,y2）代表矩形左上角和右下角点的坐标。 9 个矩形共有 3 种形状，3 scale with box areas 分别是｛128*128 256*256 512*512｝和 3 aspect ratios 分别是近似｛（1:1 1:2 2:1）｝，所以共 9 种矩形。

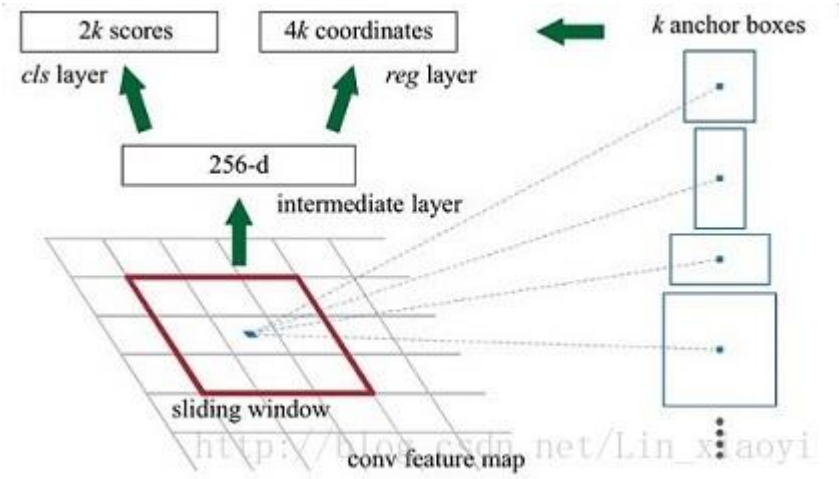
实际上通过 anchors 就引入了检测中常用的多尺度方法



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。**

anchor 的本质是什么，本质是 SPP(spatial pyramid pooling)思想的逆向。而 SPP 本身是做什么的呢，就是将不同尺寸的输入 resize 成为相同尺寸的输出。所以 SPP 的逆向就是，将相同尺寸的输出，倒推得到不同尺寸的输入。
对于每个 3×3 的窗口，作者就以这个滑动窗口的中心点对应原始图片的中心点。然后作者假定，这个 3×3 的窗口，是从原始图片通过 SPP 池化得到，而这个池化的面积及比例，就是一个个 anchors。换句话说，对于每个 3x3 窗口，作者假定它来自 9 种不同原始区域的池化，但是这些池化在原始图片中的中心点，都完全一样。这个中心点，就是刚刚提到的，3x3 窗口中心点所对应的原始图片中的中心点。如此一来，在每个窗口位置，我们都可以根据不同的长宽比例，不同的面积的 anchors，逆向推导出它所对应的原始图片的一个区域，这个区域的尺寸以及坐标，都是已知。而这个区域，就是我们想要的 proposal.接下来，每个 proposal 我们只输出 6 个参数，每个 proposal 和 ground truth 进行比较得到的前景概率和背景概率（2 个参数）对应图片上的 cls_score，由于每个 proposal 和 groundtruth 的位置及尺寸上的差异从 proposal 通过

平移缩放得到 ground truth 需要的 4 个平移缩放参数（对应图片上 bbox_pred）



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。

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

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

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(bg/fg) * 9(anchors)
    kernel_size: 1 pad: 0 stride: 1
  }
}
```

可以看到 num_output=18,也就是经过卷积的输出的图像为W×H×18 大小。这刚好对应了 feature maps 每一个点都有 9 个 anchors,同时每个 anchors 又可能是 foreground 和 background，所以这些信息都保存在 W×Hx(9x2)大小的矩阵。为啥要这样做，因为后面的 softmax 类获得 foreground anchors，也就是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：

```
"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}.";
```

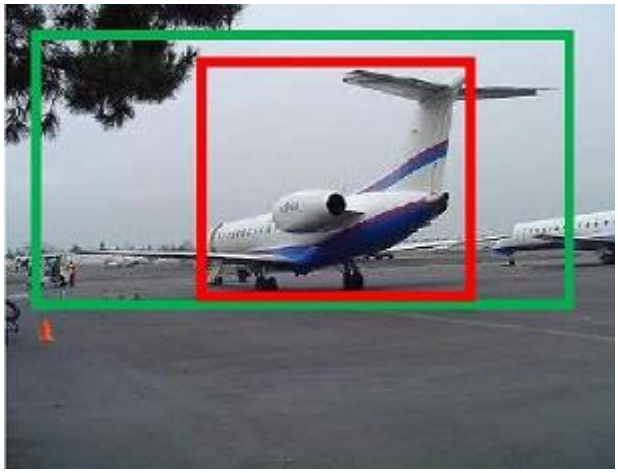
- 3
- 4

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

softmax 判断 anchors 是否属于 foreground 或者 background，这具体怎么实现我后面讲 loss 时会认真讲

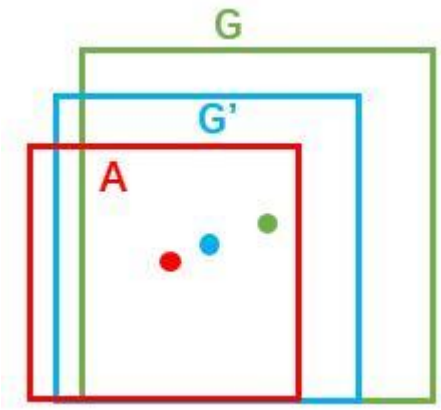
bounding box regression 原理

介绍 bounding box regression 数学模型及原理。如图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。



对于窗口我们一般用四维向量（x,y,w,h）表示，分别表示窗口的中心点坐标和宽高，如下图，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 anchorA=(Ax,Ay,Aw,Ah),GT=[Gx,Gy,Gw,Gh],寻找一种变换 F:使得 F(Ax,Ay,Aw,Ah)=(G’x,G’y,G’w,G’

h),其中（G'x,G'y,G'w,G'h）≈(Gx, Gy, Gw, Gh)。



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:

1.先做平移

G'x=Aw*dx(A)+Ax

G'y=Ah*dy(A)+Ay

2.再做缩放

G'w=Aw*exp(dw(A))

G'h=Ah*exp(dh(A))

观察上面 4 个公式发现，需要学习的是 dx(A),dy(A),dw(A),dh(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 anchorsAGT 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 Faster Rcnn 原文，平移量（tx,ty）与尺度因子（tw,th)如下：

$$t_x = (x - x_a)/w_a, \quad t_y = (y - y_a)/h_a, \quad t_w = \log(w/w_a$$

$$t_x^* = (x^* - x_a)/w_a, \quad t_y^* = (y^* - y_a)/h_a, \quad t_w^* = \log(w^*/w$$

其中 x, y, w, h 对应两组框的中心点的坐标和它的宽和高。变量 x, x_a, x^* 分别对应 predicted box ， anchor box 和 ground-truth box 的中心店横坐标（同理，y ，w ，h）我们可以这么认为 bounding box regression 就是把 anchor box 拟合到 ground-truth box。在我们的公式中，用于回归的特征是相同空间大小的在 feature maps。为了解决不同尺寸，一部分 k bounding-box-regressionors 将被学习。每一个 regressor 对应一个大小和一个比例值，而且 k regressors 不共享 weights。因此，我们可以预测不同大小的框哪怕不同大小尺度的 feature

注意，我们的 bounding box regression 是在 features pooled from arbitrarily sized regio 执行，而且我们的 regression weigths 在任意大小的 region 都是共享的

原文：Nevertheless, our method achieves bounding-box regression by a different manner from previous
feature-map-based methods [7, 5]. In [7, 5], bounding-box regression is performed on features
pooled from arbitrarily sized regions, and the regression weights are shared by all region sizes. In
our formulation, the features used for regression are of the same spatial size (n × n) on the feature
maps. To account for varying sizes, a set of k bounding-box regressors are learned. Each regressor
is responsible for one scale and one aspect ratio, and the k regressors do not share weights. As such,
it is still possible to predict boxes of various sizes even though the features are of a fixed size/scale.

接下来的问题就是如何通过线性回归获得 dx(A),dy(A),dw(A),dh(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 就是一张图片经过卷积获得的 feature map，定义为$,同时还有训练传入的 GT，即（tx,ty,tw,th）。输出的是 dx(A)，dy(A)，dw(A),dh(A)四个变换。那么目标函数可以表示为：

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

其中$(A)是对应 anchor 的 feature map 组成的向量，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上述的一个目标函数）。为了让预测值（tx,ty,tw,th)与真实值差距最小，设计损失函数：

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

函数的目标为：（smooth L1 loss)

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

对 proposals 进行 bounding box regression

了解了 bounding box regression 后，再回过头来看 RPN 网络第二条线路，如图

先来看看上图中 1×1 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 num_output=36，即经过卷积输出图像为 W×H×36，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 又有 4 个用于回归的[dx(A),dy(A),dw(A),dh(A)]变换量。

Proposal Layer

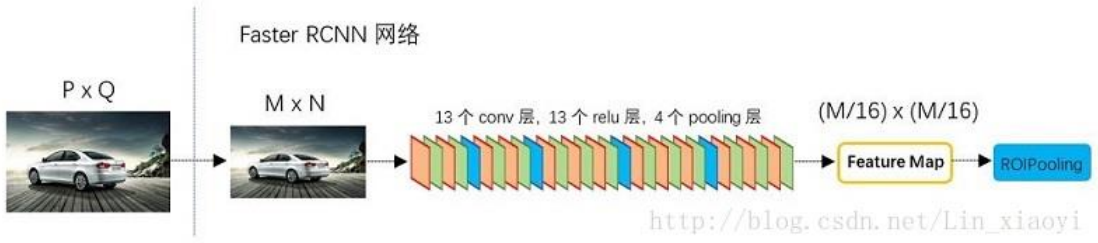
proposal layer 负责综合所有的[dx(A),dy(A),dw(A),dh(A)]变换量和 foreground anchors，计算出精准的 proposal，送入后续的 Roi pooling layer.
我们先看看 proposal layer 的 caffe prototxt 定义

```
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：fg/bg anchors 分类器结果 rpn_prob_reshape，对应的 bbox reg 的[dx(A),dy(A),dw(A),dh(A)]变换量 rpn_bbox_ped,以及 im_info,另外还有参数 feat_stride=16,这和上图对应

首先解释下 im_info，对于一幅任意大小的 P*Q 图像，传入 Fsater Rcnn 前首先 reshape 到 M*N 大小，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。



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[dx(A)，dy(A)，dw(A)，dh(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 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。
- 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。
- 进行 nms（nonmaximum suppression，非极大值抑制）（以后介绍 NMS 的原理）
- 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 网络结构总结

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

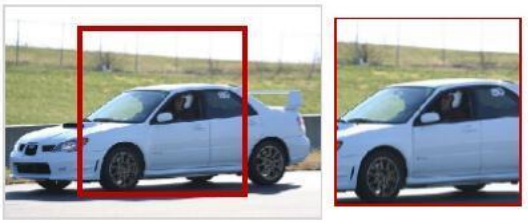
RoI pooling

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 featrue map
- RPN 输出的 proposal boxes（大小各不相同）

为何需要 ROI Pooling

先来看一个问题：对于传统的 CNN(如 alexnxt,VGG),当网络训练好后输入的图像尺寸必须是固定的，同时网络输出也是固定大小的 ovctor 或 matrix。如果输入的图像大小不定，这个问题就变得比较麻烦了。
有 2 种解决办法：
- 从图像中 crop 一部分传到网络
- 将图像 warp 成需要大小后传入网络



crop

warp

两种办法的示意图如图，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。
回忆下 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 proposal 也是大小形状各不相同，即也存在上述问题。所以 Faster RCNN 提出了 RIO Pooling 解决这个问题（RIO Pooing 确实是从 SPP 发展而来的，SPP 我们以后再讲）

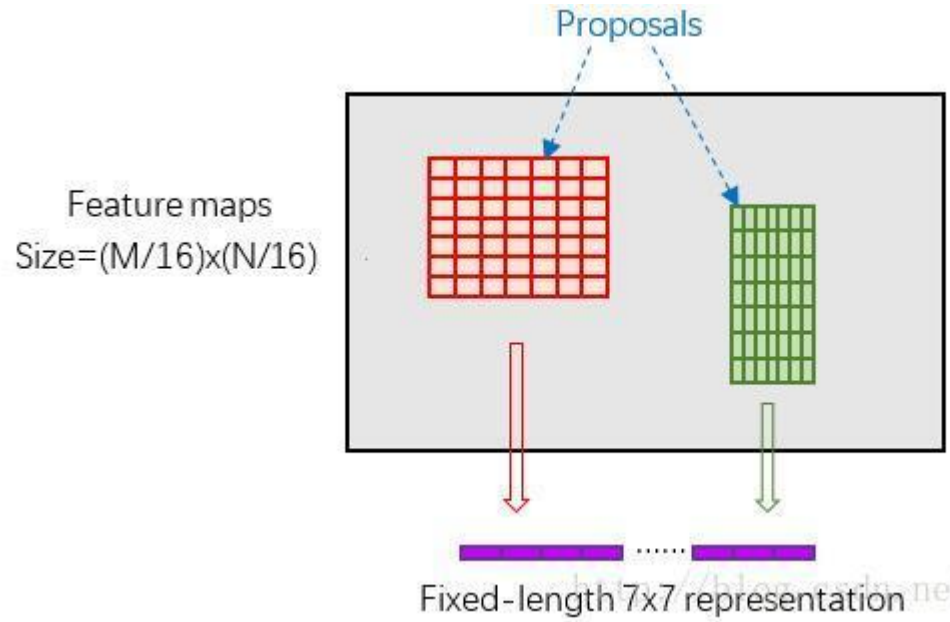
RIO Pooing 原理

分析之前我们先来看 RIO Pooling layer 的 caffe prototxt 的定义

```
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
  }
}
```

```
1
2
3
4
5
6
7
8
9
10
11
12
```

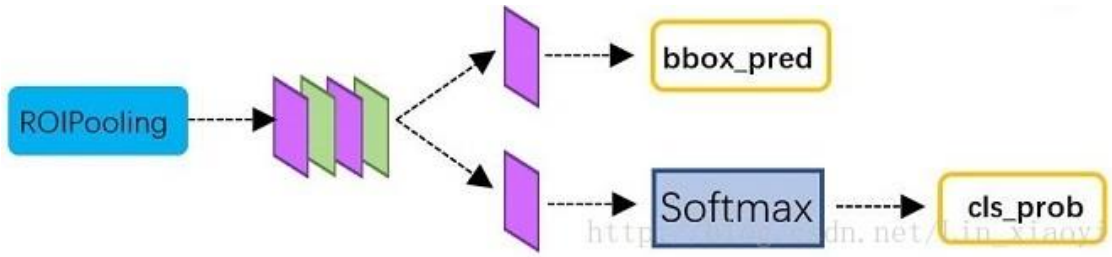
其中有新的参数 pooled_w=poold_h=7,另一个仓库 spatial_scale=1/16
RIO Pooling layerl forward 过程：在这之前有明确提到：proposal=[x1,y1,x2,y2]是对应 M*N 尺度的，所以首先使用 spatial_scale 参数将其映射回(M/16)x(N/16)大小的 feature map 尺度；之后将每个 proposal 水平方向和竖直方向都分成 7 份，对每一份都进行 max pooling 处理，这样处理后，即使大小不同的 proposal，输出的结果都是 7*7 大小的，实现了 fixed-length output（固定长度输出）。



Classification

classification 部分利用已经获得的 proposal featuer map，通过 full connect 层与 softmax 计算每个 proposal 具体属于哪个类别（如车，人等），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 部分网络结构如下：

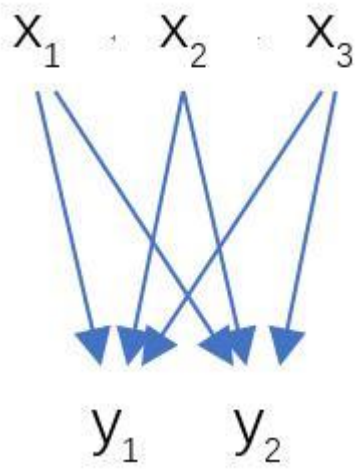


从 RIO Pooling 获取到 7*7=49 大小的 proposal feature maps 后，送入后续的网络，可以看到做了如下 2 件事：

1：通过全连接层和 softmax 对 proposal 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

这里我们来看看全连接层 InnerProduct layers，简单的示意图如下



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，即使大小是固定，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 RIO Pooling 的必要性。

Faster RCNN 训练

fastrcnn 训练方式有三种：

- 使用交替优化算法训练
- 近似联合训练
- 联合训练

交替优化训练

faster rcnn 的训练，其实是在已经训练好的 model（如 VGG_CNN_M,_1024, VGG,ZF）的基础上继续训练，实际中训练的过程分为 6 个步骤：

1. 在预训练的 model 上，训练 RPN 网络
2. 利用训练好的 RPN
3. 第一次训练 Fast-RCNN 网络
4. 第二次训练 RPN 网络
5. 再次利用步骤 4 训练好的 RPN 网络搜集 proposals
6. 第二次训练 Fast-RCNN 网络

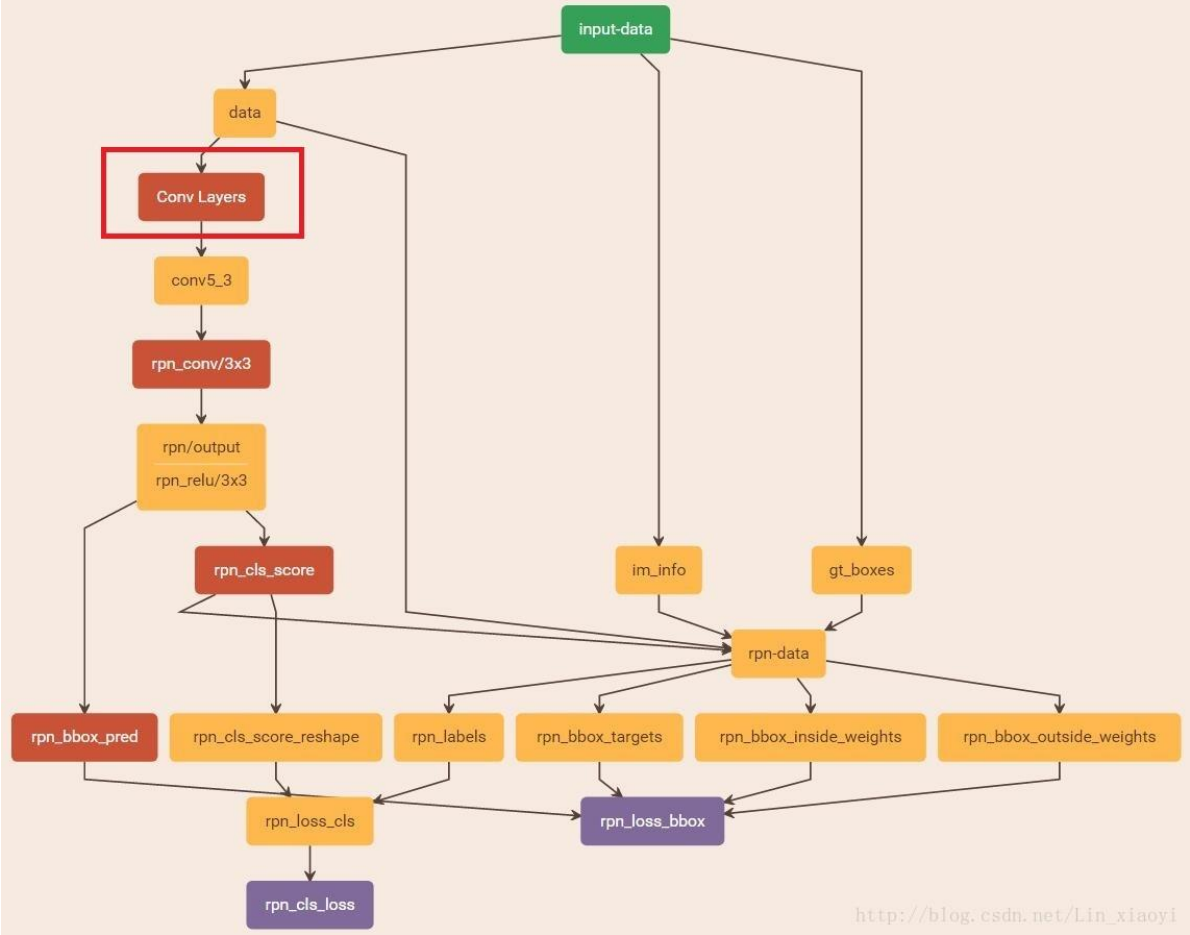
可以看到训练过程类似于一种“迭代”的过程，不过只循环了 2 次。至于只循环了 2 次的原因是应为作者提到：”A similar alternating training can be run for more iterations, but we have observed negligible improvements”，即循环更多次没有提升了。

近似联合训练

直接一下子训练完成

训练 RPN 网络

在该步骤中首先读取预训练 model，开始进行迭代训练



与检测网络类似，依然使用 conv layers 提取 feature map.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,pi 表示 foreground softmax predict 概率，p_i^* 代表对应的 GT predict 概率（即当第 i 个 anchors 与 GT 间 IoU>0.7,认为该

anchor 是 foreground, p_i^*=1，反之 IOU<0.3 时，认为该 anchors 是 background, p_i^*=0；至于那些 0.3 小于 IOU<0.7 的 anchors 则不参与训练，一般一张图片取 256 个 anchors,一般 bg 和 fg＝1;1）；t 代表 predict bounding box, t^*代表对应的 foreground anchors 对应的 GT box。可以看到，整个 LOSS 分为 2 个部分：

1．cls loss,即 rpn_cls_loss 层计算的 softmax loss,用于分类的 anchors 为 fg 与 bg 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 pi*，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，Ncls 和 Nreg 差距过大，用参数 λ 平衡二者（如 Ncls=256，Nreg=2400 时设置 λ=10），使总的网络 Loss 计算过程中能够均匀考虑两种 Loss。这里比较重要是 Lreg 使用的 soomth L1 loss，计算公式如下：

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

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. 对与 rpn_loss_cls，输入的 rpn_cls_score_reshape 和 rpn_labels 分别对应 p 与 p*，Ncls 参数隐含在 p 与 p*的 caffe blob 的大小中

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！

通过训练好的 RPN 网络收集 proposals

进在该步骤中，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。

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，如图篮框
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。