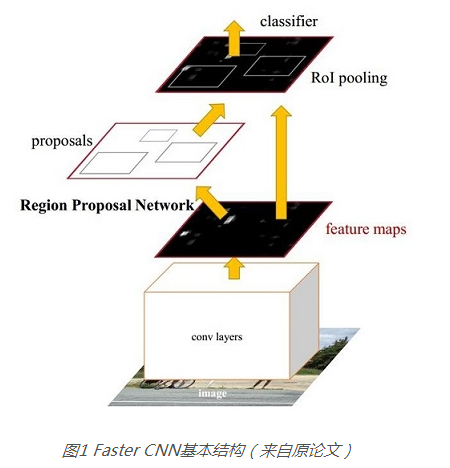
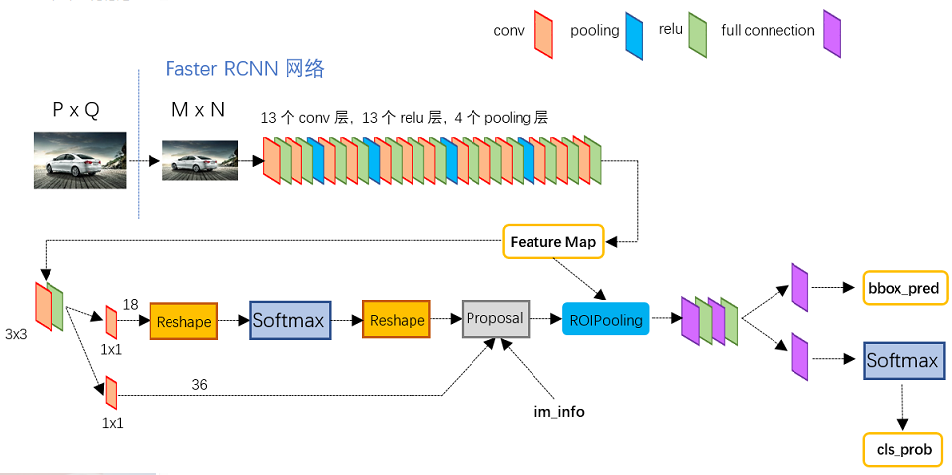
经过RCNN和Fast RCNN的积淀，Ross B. Girshick在2016年提出了新的Faster RCNN，在结构上，Faster RCN已经将特征抽取(feature extraction)，proposal提取，bounding box regression(rect refine)，classification都整合在了一个网络中，使得综合性能有较大提高，在检测速度方面尤为明显。



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。

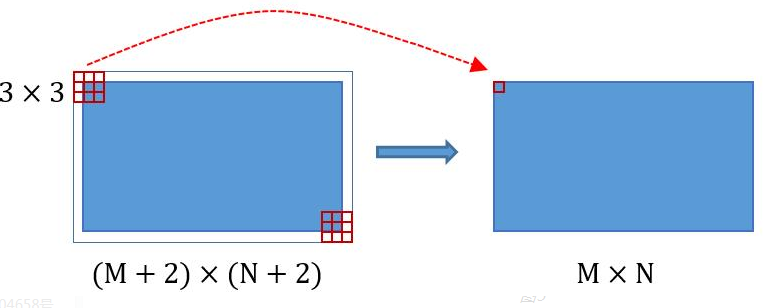
图2 faster\_rcnn\_test.pt网络结构

**1 Conv layers**

缩进Conv layers包含了conv，pooling，relu三种层。以[**python**](http://lib.csdn.net/base/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

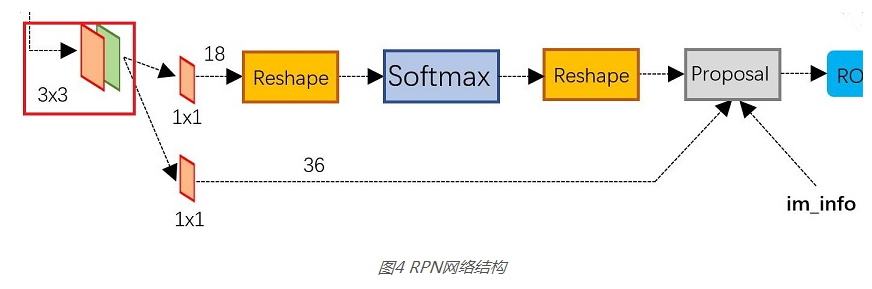
为何重要？在Faster RCNN Conv layers中对所有的卷积都做了扩边处理（pad=1，即填充一圈0），导致原图变为(M+2)x(N+2)大小，再做3x3卷积后输出MxN。正是这种设置，导致Conv layers中的conv层不改变输入和输出矩阵大小。如图3：



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。

缩进那么，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 Proposal Networks(RPN)



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

缩进在介绍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：

### 

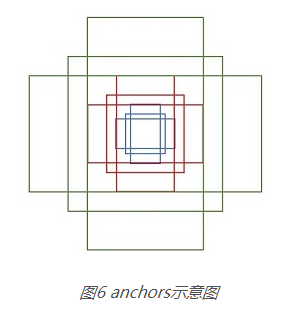
如图5，输入图像layer m-1有4个通道，同时有2个卷积核w1和w2。对于卷积核w1，先在输入图像4个通道分别作卷积，再将4个通道结果加起来得到w1的卷积输出；卷积核w2类似。所以对于某个卷积层，无论输入图像有多少个通道，输出图像通道数总是等于卷积核数量！

缩进对多通道图像做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

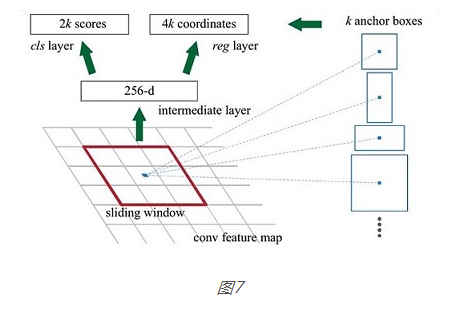
提到RPN网络，就不能不说anchors。所谓anchors，实际上就是一组由rpn/generate\_anchors.py生成的矩形。

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。



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](https://www.baidu.com/s?wd=%E5%86%8D%E5%9B%9E%E5%A4%B4&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)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。



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，训练程序会选取256个合适的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

缩进一副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：



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）。

缩进那么为何要在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]**

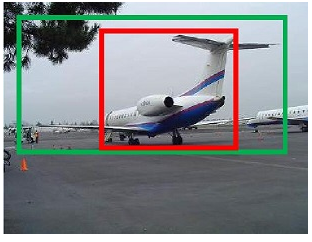
对应至上面的保存bg/fg anchors的矩阵，其在caffe blob中的存储形式为[1, 2\*9, H, W]。而在softmax分类时需要进行fg/bg二分类，所以reshape layer会将其变为[1, 2, 9\*H, W]大小，即单独“腾空”出来一个维度以便softmax分类，之后再reshape回复原状。贴一段caffe softmax\_loss\_layer.cpp的reshape函数的解释，非常精辟：

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

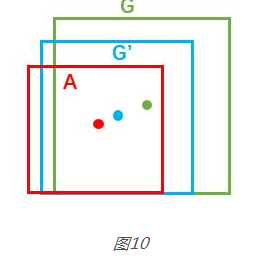
综上所述，RPN网络中利用anchors和softmax初步提取出foreground anchors作为候选区域。

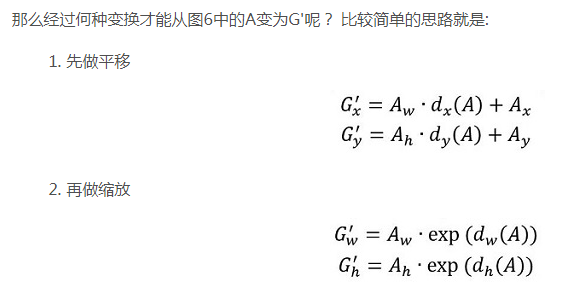
### 2.4 bounding box regression原理

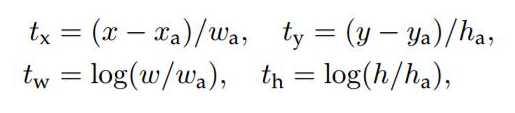
缩进介绍bounding box regression数学模型及原理。如图9所示绿色框为飞机的Ground Truth(GT)，红色为提取的foreground anchors，那么即便红色的框被分类器识别为飞机，但是由于红色的框定位不准，这张图相当于没有正确的检测出飞机。所以我们希望采用一种方法对红色的框进行微调，使得foreground anchors和GT更加接近。

图9

进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A=(Ax, Ay, Aw, Ah)，寻找一种映射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)



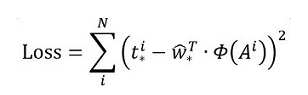


观察上面4个公式发现，需要学习的是dx(A)，dy(A)，dw(A)，dh(A)这四个变换。当输入的anchor与GT相差较小时，可以认为这种变换是一种线性变换， 那么就可以用线性回归来建模对窗口进行微调（注意，只有当anchors和GT比较接近时，才能使用线性回归模型，否则就是复杂的非线性问题了）。对应于Faster RCNN原文，平移量(tx, ty)与尺度因子(tw, th)如下：

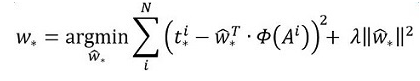
接下来的问题就是如何通过线性回归获得dx(A)，dy(A)，dw(A)，dh(A)了。线性回归就是给定输入的特征向量X, 学习一组参数W, 使得经过线性回归后的值跟真实值Y（即GT）非常接近，即Y=WX。对于该问题，输入X是一张经过num\_output=1的1x1卷积获得的feature map，定义为Φ；同时还有训练传入的GT，即(tx, ty, tw, th)。输出是dx(A)，dy(A)，dw(A)，dh(A)四个变换。那么目标函数可以表示为：



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)与真实值最小，得到损失函数：



函数优化目标为：



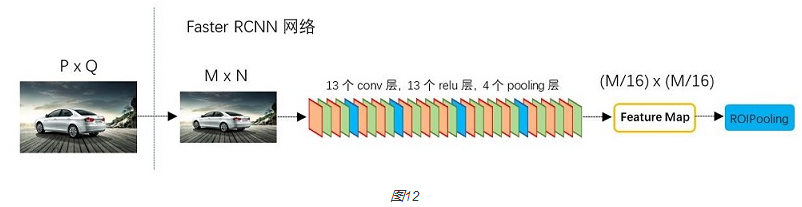
可以看到其num\_output=36，即经过该卷积输出图像为WxHx36，在caffe blob存储为[1, 36, H, W]。与上文中fg/bg anchors存储为[1, 18, H, W]类似，这里相当于feature maps每个点都有9个anchors，每个anchors又都有4个用于回归的[dx(A)，dy(A)，dw(A)，dh(A)]变换量。利用上面的的计算公式即可从foreground anchors回归出proposals。

### 2.6 Proposal Layer

缩进Proposal Layer负责综合所有[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RoI Pooling Layer。

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[dx(A)，dy(A)，dw(A)，dh(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。

缩进首先解释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。所有这些数值都是为了将proposal映射回原图而设置的，如图12，毕竟检测就是为了在原图上画一个框而已~



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

1. 再次生成anchors，并对所有的anchors做bbox reg位置回归（注意这里的anchors生成顺序和之前是即完全一致的）
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feat\_stride和im\_info将anchors映射回原图，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**

缩进而RoI Pooling层则负责收集proposal，并计算出proposal feature maps，送入后续网络。从图3中可以看到Rol 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

缩进先来看一个问题：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：

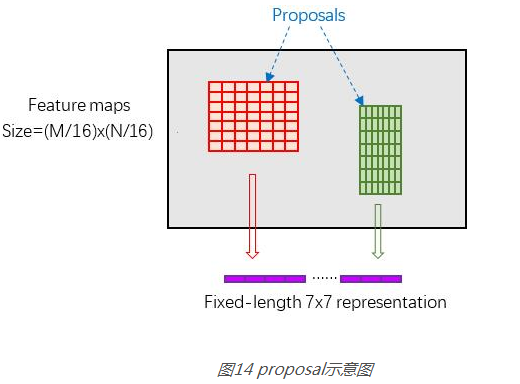
1. 从图像中crop一部分传入网络
2. 将图像warp成需要的大小后传入网络



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）。

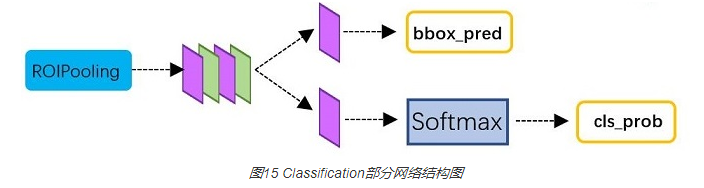
### 3.2 RoI Pooling原理

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MxN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。



**4 Classification**

缩进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。



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

**5 Faster RCNN训练**

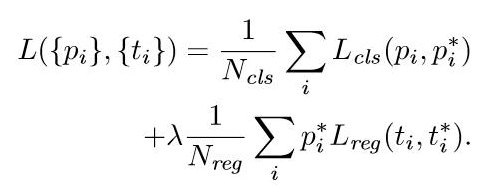
缩进Faster CNN的训练，是在已经训练好的model（如VGG\_CNN\_M\_1024，VGG，ZF）的基础上继续进行训练。实际中训练过程分为6个步骤：

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。

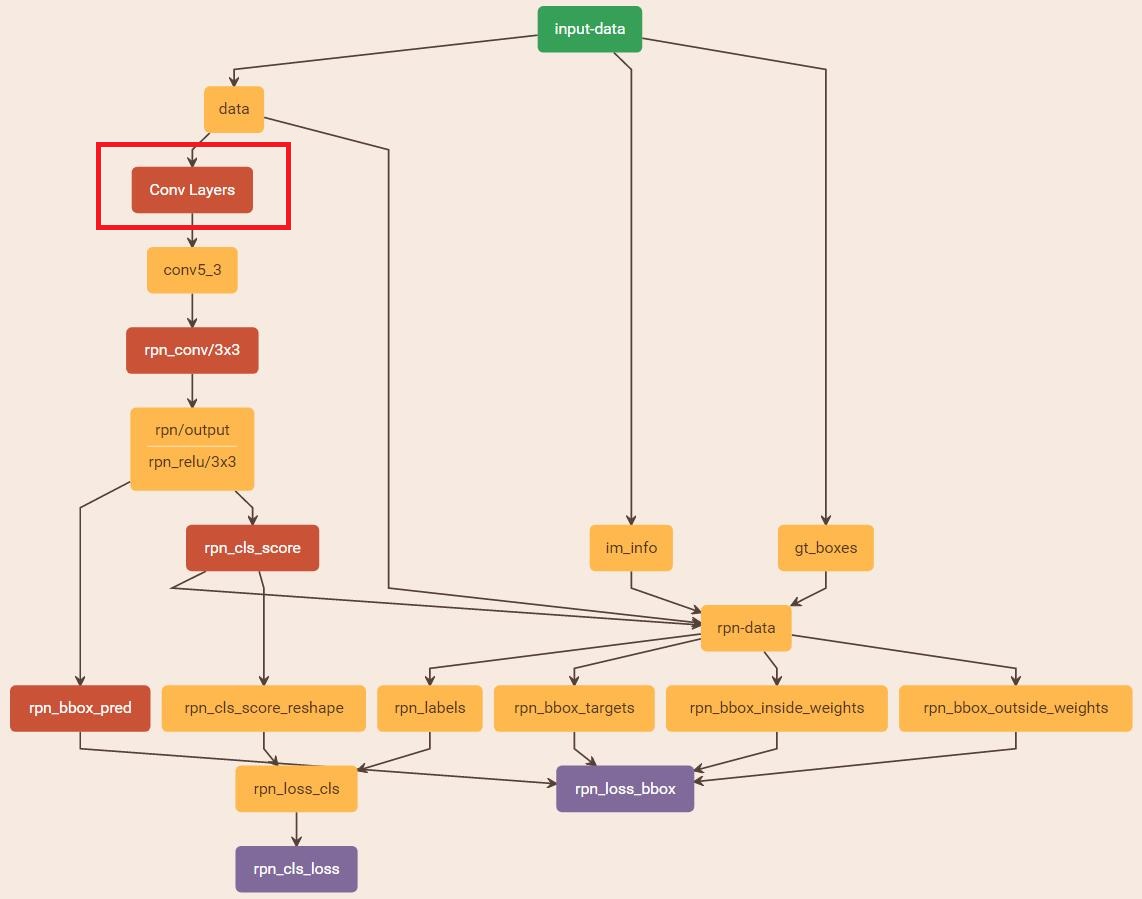
### 5.1 训练RPN网络

缩进在该步骤中，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。



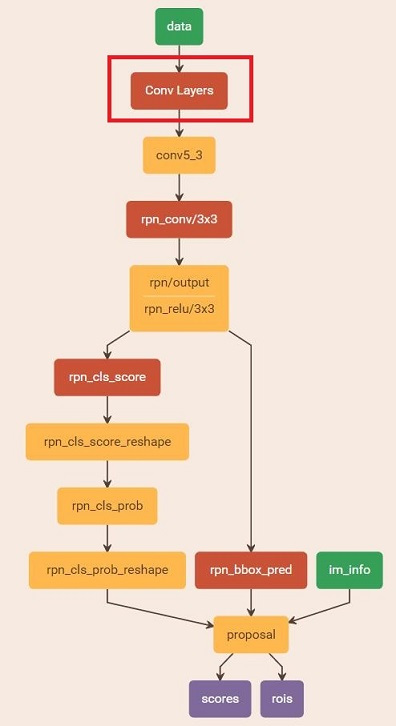
上述公式中，i表示anchors index，pi表示foreground softmax predict概率，pi\*代表对应的GT predict概率（即当第i个anchor与GT间IoU>0.7，认为是该anchor是foreground，pi\*=1；反之IoU<0.3时，认为是该anchor是background，pi\*=0；至于那些0.3<IoU<0.7的anchor则不参与训练）；t代表predict bounding box，t\*代表对应foreground anchor对应的GT box。可以看到，整个Loss分为2部分：

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）。



### 5.2 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。

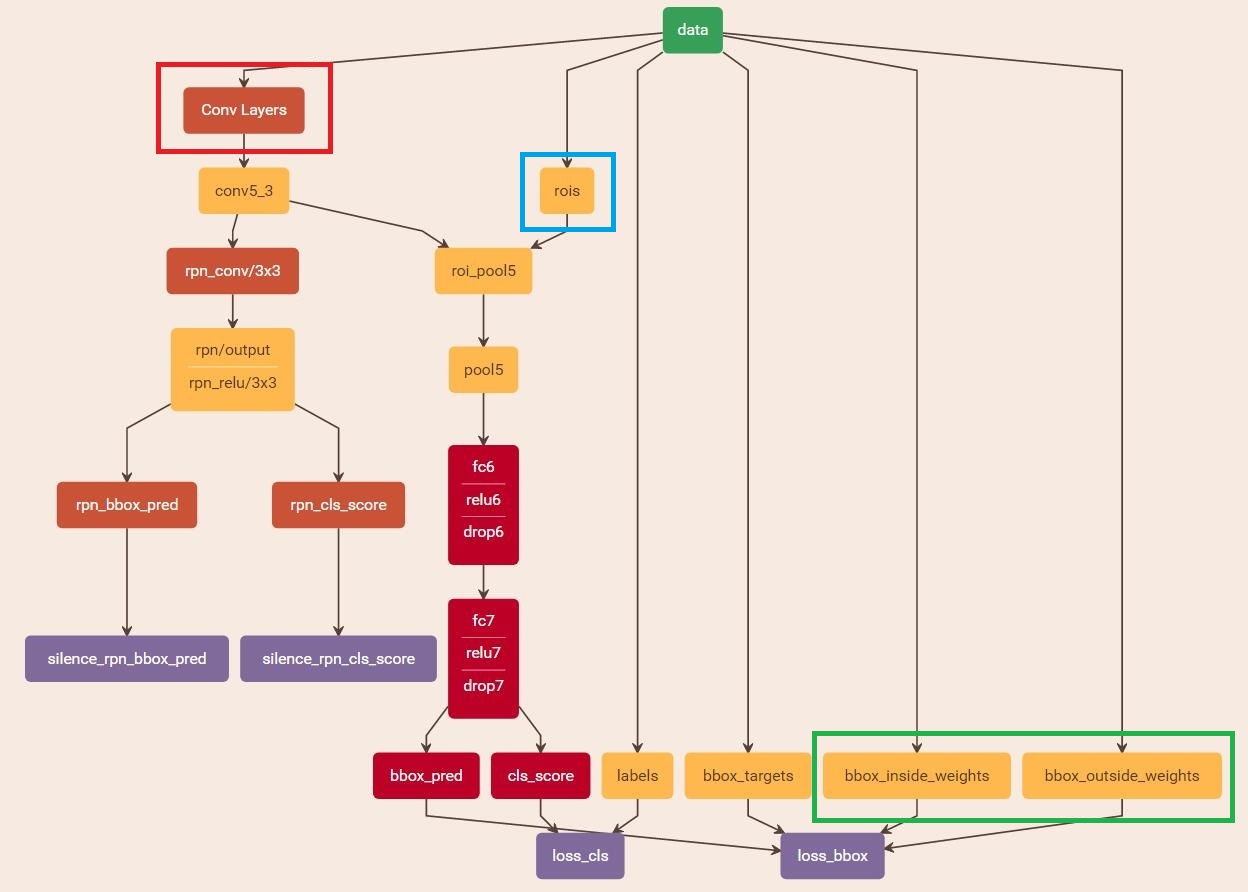


### 5.3 训练Fast RCNN网络

缩进读取之前保存的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。



这个画图工具

Netscope:支持Caffe的神经网络结构在线可视化工具  
https://blog.csdn.net/10km/article/details/52713027