**目标检测算法:Faster R-CNN发展历程及介绍（jimmybhb）**

1. **Faster R-CNN算法心得来源**：

对多目标检测算法Faster R-CNN 的心得来源学习华为云【ModelArts-Lab AI实战营】第五期：物体检测（ll），通过该次学习，从而了解**Faster R-CNN**网络结构及算法。

1. **网络结构组成**:RPN+CNN+ROI三部分组成，其中：

**RPN**：为Region Proposal Network简写，中文意思为：“提取候选框”[请点击...](https://www.cnblogs.com/Terrypython/p/10584384.html)

**CNN**：卷积神经网络（深度学习，可参考之前学习的常用卷积神经网络学习算法：如VGG16（[请点击...](https://blog.csdn.net/amcle/article/details/79165348)）模型），[CNN详情介绍请点击...](https://www.cnblogs.com/charlotte77/p/7759802.html)

**ROI**：为Region of Interest缩写,中文意思为：“特征图上的框（或感兴趣区域）”

1. **算法思路**：

* 候选区域生成
  + - 特征提取
* 分类，位置精修

1. **目标检测（object detection）算法分类**：

根据目标检测发展和神经网络模型的发展，对目标检测算法主要分三个阶段，分别如下：

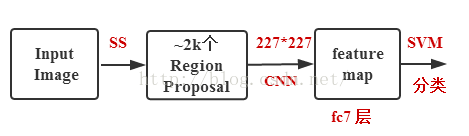
1. **传统的目标检测算法**：

Cascade + HOG/DPM + Haar/SVM以及上述方法的诸多改进、优化，此处不做讲解，大家可以去网上查找相关文章

1. **候选区域/框 + 深度学习分类**：通过提取候选区域，并对相应区域进行以深度学习方法为主的分类的方案，常用算法有如下：

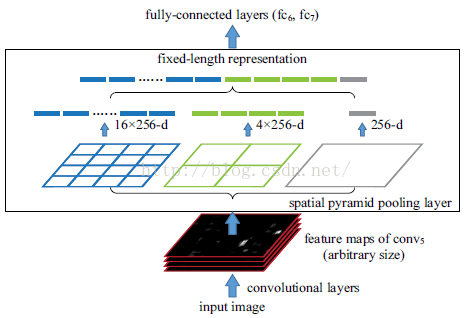
* R-CNN（Selective Search + CNN + SVM）

R-CNN目标检测流程如下：



[R-CNN详情介绍请点击...](http://blog.csdn.net/shenxiaolu1984/article/details/51066975)

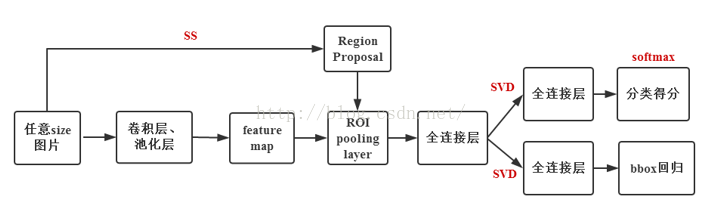
* + - SPP-net（空间金字塔池化ROI Pooling）



[SPP-net详情介绍请点击...](https://blog.csdn.net/qq_26898461/article/details/50424240)

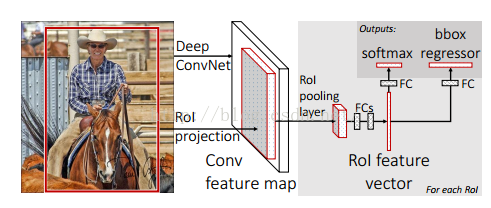
* Fast R-CNN（Selective Search + CNN + ROI）

Fast R-CNN目标检测流程如下:



注意：Fast R-CNN的RegionProposal是在feature map之后做的，这样可以不用对所有的区域进行单独的CNN Forward步骤。

Fast R-CNN框架如下图：

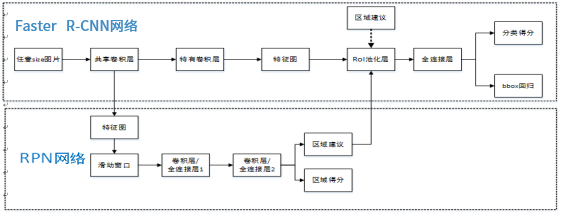


Fast R-CNN框架与R-CNN有两处不同：

① 最后一个卷积层后加了一个ROI pooling layer；

② 损失函数使用了multi-task loss（多任务损失）函数，将边框回归直接加到CNN网络中训练。分类直接用softmax替代R-CNN用的SVM进行分类。[详情介绍请点击...](http://blog.csdn.net/shenxiaolu1984/article/details/51036677)

* + - Faster R-CNN（RPN + CNN + ROI）等系列方法

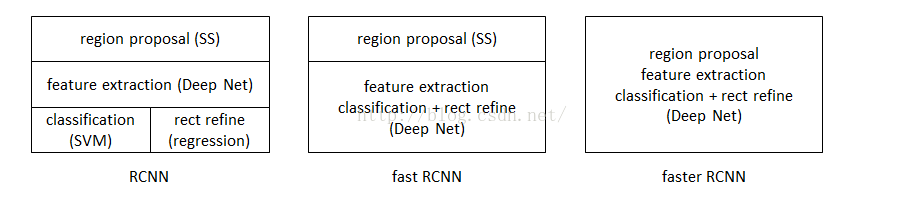


1. **基于深度学习的回归方法**：YOLO/SSD/DenseBox 等方法；以及最近出现的结合RNN算法的RRC detection；结合DPM的Deformable CNN等

此处不做过多讲解，如要熟悉请网上查询相关资料

在华为云【ModelArts-Lab AI实战营】第四期：物体检测（l）中采用YOLO模型；在华为云【ModelArts-Lab AI实战营】第五期：物体检测（ll）中采用Faster R-CNN模型，从目标检测发展历程算法：R-CNN、Fast R-CNN、Faster R-CNN可以看出，对候选区域采用高效先进的RPN网络替代了Select Search算法。

其中：**R-CNN、Fast R-CNN、Faster R-CNN三者之间的关系如下图**:



|  |  |  |  |
| --- | --- | --- | --- |
|  | **使用方法** | **缺点** | **改进** |
| R-CNN  (Region-based Convolutional  Neural Networks) | 1、SS提取RP；  2、CNN提取特征；  3、SVM分类；  4、BB盒回归。 | 1、训练步骤繁琐（微调网络+训练SVM+训练bbox）；  2、训练、[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库)均速度慢 ；  3、训练占空间 | 1、从DPM HSC的34.3%直接提升到了66%（mAP）；  2、引入RP+CNN |
| Fast R-CNN  (Fast Region-based Convolutional  Neural Networks) | 1、SS提取RP；  2、CNN提取特征；  3、softmax分类；  4、多任务损失函数边框回归。 | 1、依旧用SS提取RP(耗时2-3s，特征提取耗时0.32s)；  2、无法满足实时应用，没有真正实现端到端训练测试；  3、利用了GPU，但是区域建议方法是在CPU上实现的。 | 1、由66.9%提升到70%；  2、每张图像耗时约为3s。 |
| Faster R-CNN  (Fast Region-based Convolutional  Neural Networks) | 1、RPN提取RP；  2、CNN提取特征；  3、softmax分类；  4、多任务损失函数边框回归。 | 1、还是无法达到实时检测目标；  2、获取region proposal，再对每个proposal分类计算量还是比较大。 | 1、提高了检测精度和速度；  2、真正实现端到端的目标检测框架；  3、生成建议框仅需约10ms。 |

1. **Faster R-CNN之SPP(空间金字塔池化)原理详解**：
2. **SPP是什么?**

SPP是Spatial Pyramid Pooling的简写，即空间金字塔池化

1. **SPP在Faster R-CNN中的作用是什么**？

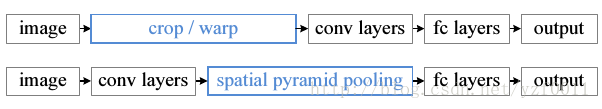
SPP主要应用在**Faster R-CNN网络模型中的**ROI Pooling层，其作用是将RPN网络模型中输出的不同尺寸大小的ROIs特征通过ROI Pooling层后转变成固定统一大小的输出，方便后期模型全连接层的分类和回归。

在一般的CNN结构中，在卷积层后面通常连接着全连接。而全连接层的特征数是固定的，所以在网络输入的时候，会固定输入的大小(fixed-size)。但在现实中，我们的输入的图像尺寸总是不能满足输入时要求的大小。然而通常的手法就是裁剪(crop)和拉伸(warp)。



这样做总是不好的：图像的纵横比(ratio aspect) 和 输入图像的尺寸是被改变的。这样就会扭曲原始的图像。而Kaiming He在这里提出了一个SPP(Spatial Pyramid Pooling)层能很好的解决这样的问题， 但SPP通常连接在最后一层卷积层。

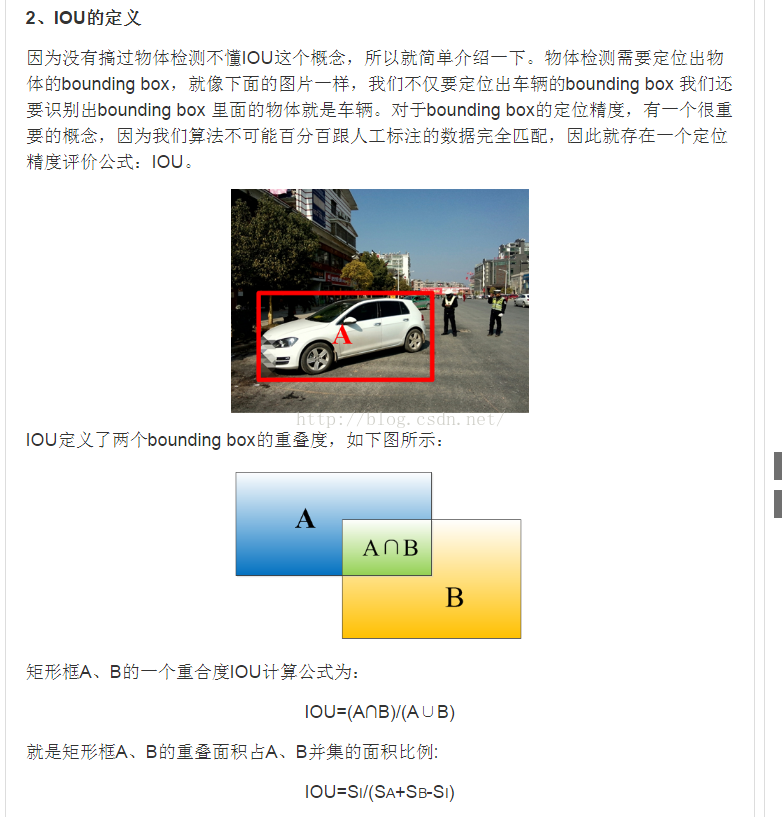
**图像剪切/拉伸处理方式 VS SPP 图对比如下：**



具体详细细节[请点击这里...](https://blog.csdn.net/qq_26898461/article/details/50424240)

1. **IoU的定义**：

在Faster R-CNN算法中，采用RPN(Region Proposal,即候选区域）网络预先找出图中目标可能出现的位置，通过利用图像中的**纹理、边缘、颜色**等信息，保证在选取较少窗口（几千个甚至几百个）的情况下保持较高的召回率（IoU，Intersection-over-Union）



Region Proposal方法比传统的滑动窗口方法获取的质量要更高。比较常用的Region Proposal方法有：SelectiveSearch(SS，选择性搜索)、Edge Boxes（EB）。

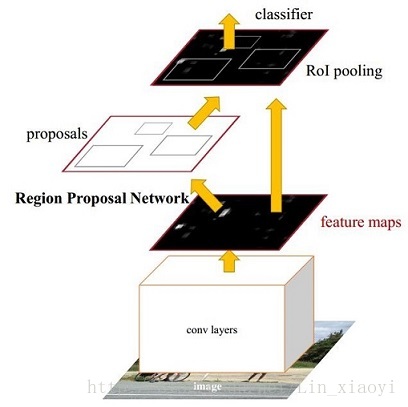
基于Region Proposal目标检测**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库)**的步骤如下：

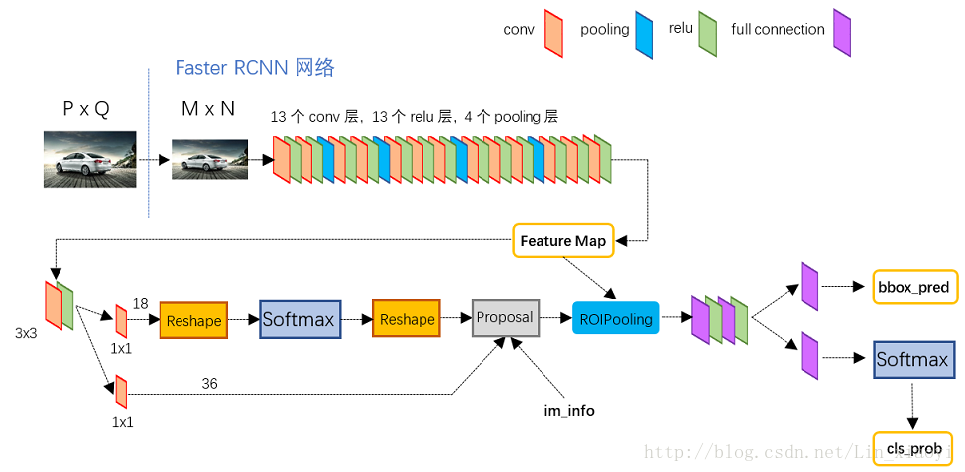


CNN方法[请点击...](http://blog.csdn.net/qq_17448289/article/details/52850223)，边框回归（Bouding Box Regression）：是对RegionProposal进行纠正的线性回归**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库)**，目的是为了让Region Proposal提取到的窗口与目标窗口（Ground Truth）更加吻合

**以下重点说说Faster R-CNN框架原理解析([原理来源...](https://blog.csdn.net/Lin_xiaoyi/article/details/78214874))**

1. **Faster R-CNN原理主框架图:**





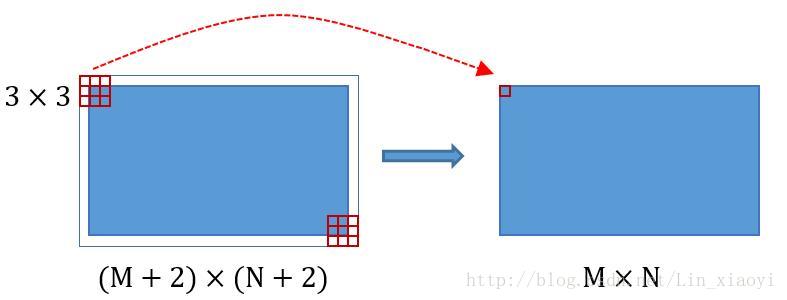
1. **Faster R-CNN算法步骤:**
2. **Conv layers**.作为一种cnn网络目标检测的方法，faster\_rcnn首先使用一组基础conv+relu+pooling层提取image的feture map。该feature map被共享用于后续的RPN层和全连接层。
3. **Region Proposal Networks.RPN**层是faster-rcnn最大的亮点，RPN网络用于生成region proposcals.该层通过softmax判断anchors属于foreground或者background，再利用box regression修正anchors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）
4. **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。
5. **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
6. **Faster R-CNN原理详解：**
7. **Conv　layer**

在input-data层时，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

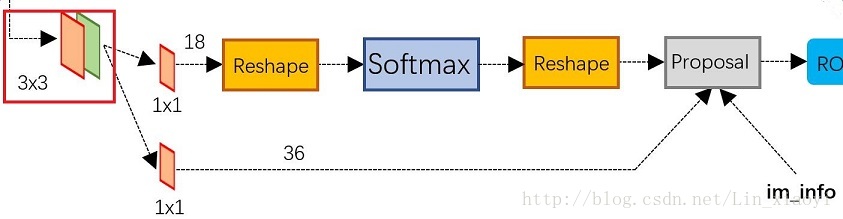


1. **RPN(Region Propocal Networks区域生成网络)**

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。

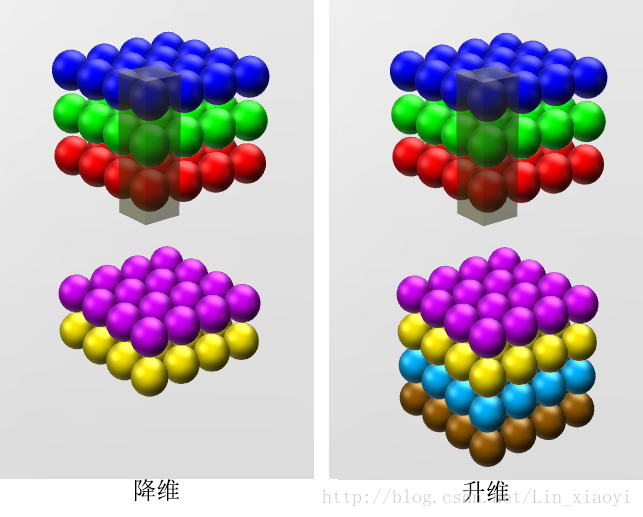
经典的检测方法生成检测框都非常耗时，如OpenCV adaboost使用滑动窗口+图像金字塔生成检测框；或如RCNN使用SS(Selective Search)方法生成检测框。而Faster RCNN则抛弃了传统的滑动窗口和SS方法，直接使用RPN生成检测框，这也是Faster RCNN的巨大优势，能极大提升检测框的生成速度。



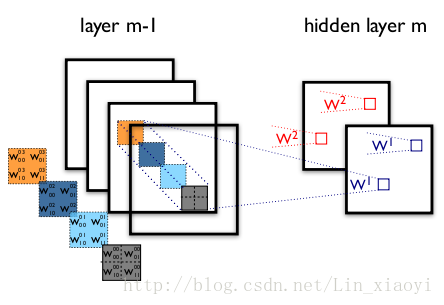
上图中展示了RPN网络的具体结构，可以看到，feature map 经过一个3×3卷积核卷积后分成了两条线，上面一条通过softmax对anchors分类获得foreground和background（检测目标是foregrounnd），因为是２分类，所以它的维度是2k　scores。下面那条线是用于计算anchors的bounding box regression的偏移量，以获得精确的proposal。它的维度是4k coordinates。而最后的proposcal层则负责综合foreground anchors和bounding box regression偏移量获取proposal，同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

**1\*1;3\*3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！

**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.]

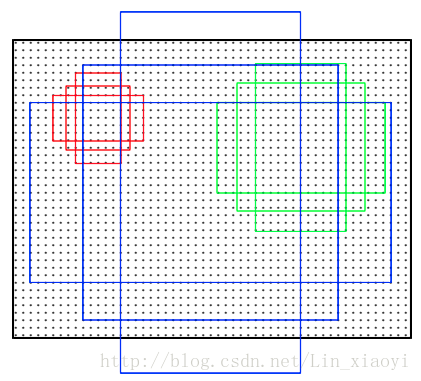
[-248. -248. 263. 263.]

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

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

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

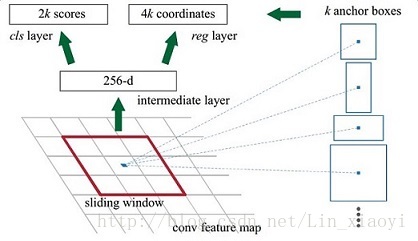
其中每行的４个值（x1,y1,x2,y2）代表矩形左上角和右下角点的坐标。９个矩形共有３种形状，3 scale with box areas分别是｛128128 256256 512\*512｝和 3 aspect ratios分别是近似｛（1:1 1:2 2:1）｝，所以共９种矩形。实际上通过anchors就引入了检测中常用的多尺度方法



**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, 29, 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, 9H, 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), "

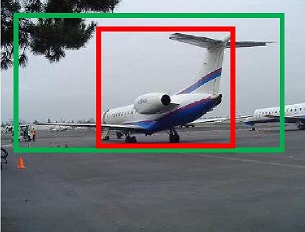
"label count (number of labels) must be N\*H\*W, "

"with integer values in {0, 1, ..., C-1}.";

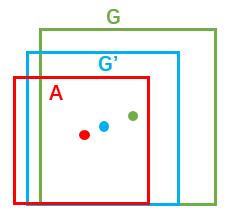
综上所述，RPN网络中利用anchors和softmax初步提取了foreground anchors作为候选区域。softmax判断anchors是否属于foreground或者background，这具体怎么实现我后面讲loss时会认真讲

**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=Awdx(A)+Ax

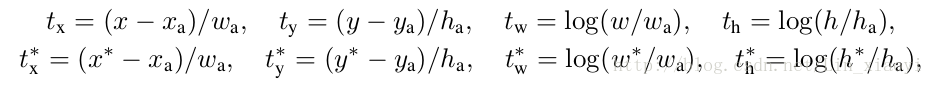
　　　　　　　　　　　　　　G’y=Ahdy(A)+Ay

2.再做缩放

　　　　　　　　　　　　　　G’w=Awexp(dw(A))

　　　　　　　　　　　　　　G’h=Ahexp(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：



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,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

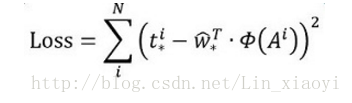
our formulation, the features used for regression are of the same spatial size (n × n) on the feature

maps. To account for varying sizes, a set of k bounding-box regressors are learned. Each regressor

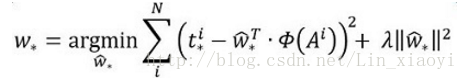
is responsible for one scale and one aspect ratio, and the k regressors do not share weights. As such,

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：![这里写图片描述](http://img.blog.csdn.net/20171013001736522?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvTGluX3hpYW95aQ==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)其中 ,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：![这里写图片描述](https://img-blog.csdn.net/20171013001736522?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvTGluX3hpYW95aQ==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)其中,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：![这里写图片描述](http://img.blog.csdn.net/20171013001736522?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvTGluX3hpYW95aQ==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)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：



函数的目标为：（smooth L1 loss)



**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。

1. **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'

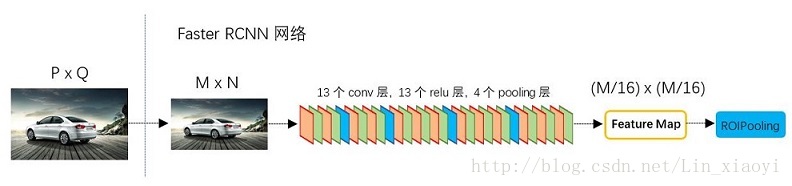
param\_str: "'feat\_stride': 16"

}

}

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PQ图像，传入Fsater Rcnn前首先reshape到MN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：

1. 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）
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）（以后介绍NMS的原理）
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网络结构总结:  
生成anchors -> softmax分类器提取fg anchors -> bbox reg回归fg anchors -> Proposal Layer生成proposals**

1. **RoI Pooling层:**

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Rol pooling层有2个输入：

1. 原始的featrue map
2. 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



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ROI Pooling解决这个问题（ROI Pooing确实是从SPP发展而来的，SPP我们以后再讲）

**ROI Pooling原理**:

分析之前我们先来看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

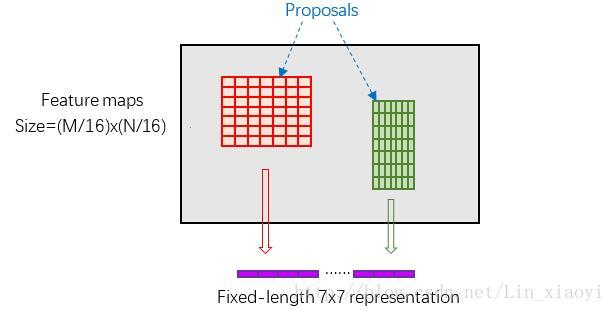
spatial\_scale: 0.0625 # 1/16

}

}

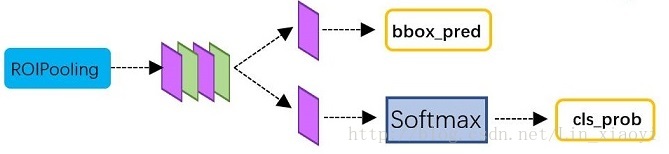
其中有新的参数pooled\_w=poold\_h=7,另一个仓库spatial\_scale=1/16

ROI Pooling layerl forward过程：在这之前有明确提到：proposal=[x1,y1,x2,y2]是对应MN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77大小的，实现了fixed-length output（固定长度输出）。



1. **分类(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：

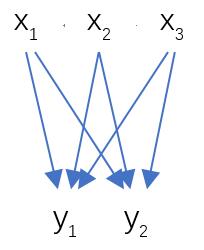


从ROI Pooling获取到7\*7=49大小的proposal feature maps后，送入后续的网络，可以看到做了如下２件事：

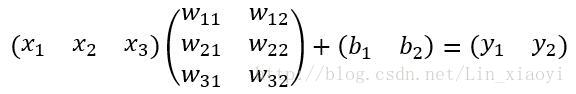
１：通过全连接层和softmax对proposal进行分类，这实际上已经是识别的范畴了

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:



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ROI Pooling的必要性。

1. **Faster R-CNN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

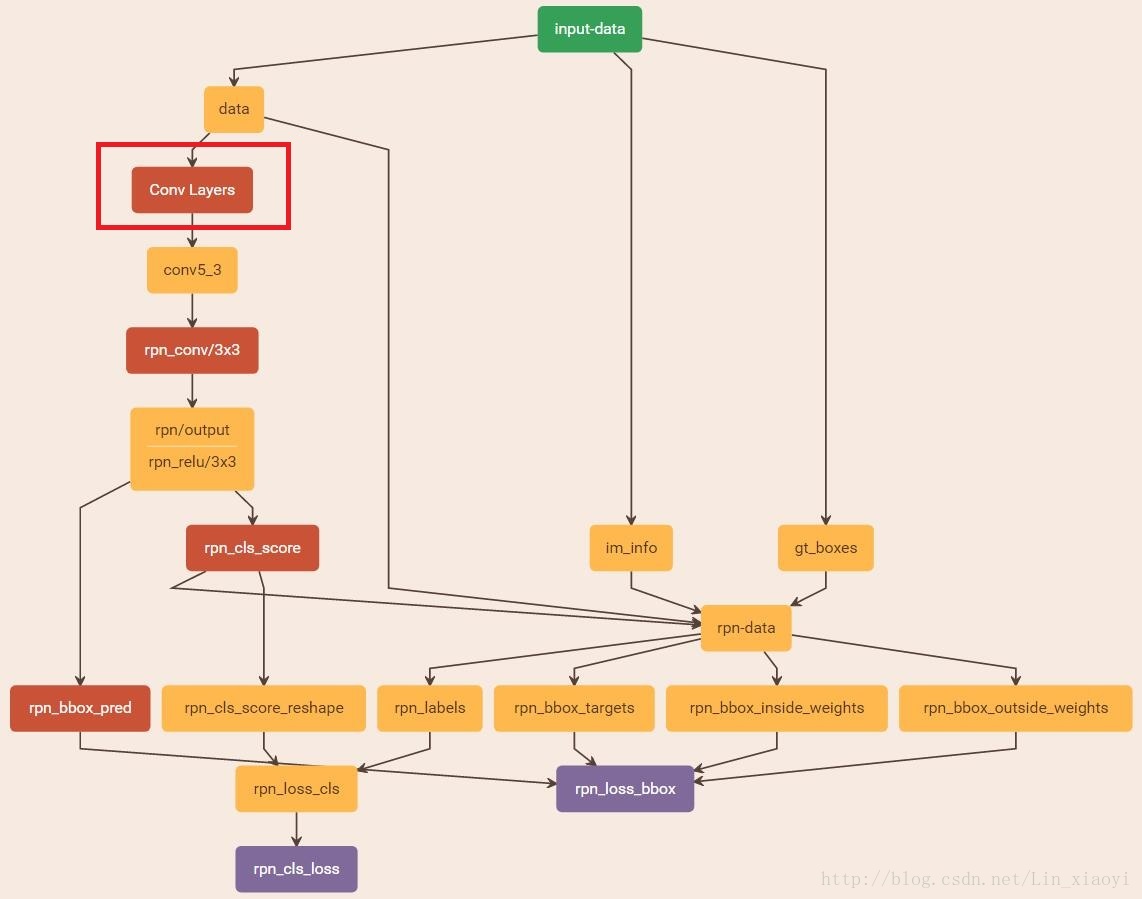
注:可以看到训练过程类似于一种“迭代”的过程，不过只循环了2次。至于只循环了2次的原因是应为作者提到：“A similar alternating training can be run for more iterations, but we have observed negligible improvements”，即循环更多次没有提升了。

****近似联合训练****

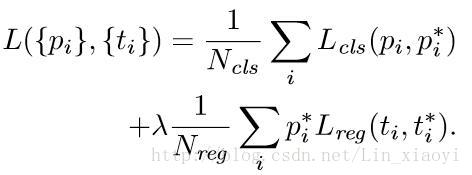
直接一下子训练完成

**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：

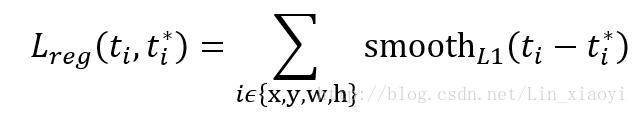


上述公式中，i表示anchors index,pi表示foreground softmax predict概率，代表对应的GT predict概率（即当第i个anchors与GT间IoU>0.7,认为该anchor是foreground,=1，反之IOU<0.3时，认为该anchors是background,=0；至于那些0.3小于IOU<0.7的anchors则不参与训练，一般一张图片取256个anchors,一般bg和fg＝1;1）；t代表predict bounding box,代表对应的foreground anchors对应的GT box。可以看到，整个LOSS分为２个部分：

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：



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 F-CNN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：

将提取的proposals作为rois传入网络，如图篮框

将foreground probability作为bbox\_inside\_weights传入网络，如图绿框

通过caffe blob大小对比，计算出bbox\_outside\_weights（即λ），如图绿框

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

