<=============================================================================>

**=====>Title**

DSOD: Learning Deeply Supervised Object Detectors from Scratch

**=====>Main Contributions / New Opinions**

1.未使用ImageNet模型进行参数初始化，而是直接利用Pascal VOC 2012数据集进行训练，得到的效果媲美其他四类检测方法

**=====>Key Points**

1.Proposal-free

Faster R-CNN以及R-FCN方法在VOC数据集上直接初始化训练，会失败。作者认为这与ROI Pooling相关，因为ROI Pooling需

要输入region proposal，而proposal-free类型的方法（SSD，YOLO）可以

2.Deep Supervision

参考DenseNet，利用skip connections实现supervised signals传递。Transition w/o Pooling Layer用这个层来增加dense

blocks数量。原来的DenseNet的dense blocks数量是固定的

3.Stem Block

Stem Block受Inception-v3和v4的启发，定义Stem Block为三个3×3卷积层和一个2×2最大池化层，发现这么设计可以提升性能

4.Dense Prediction Structure

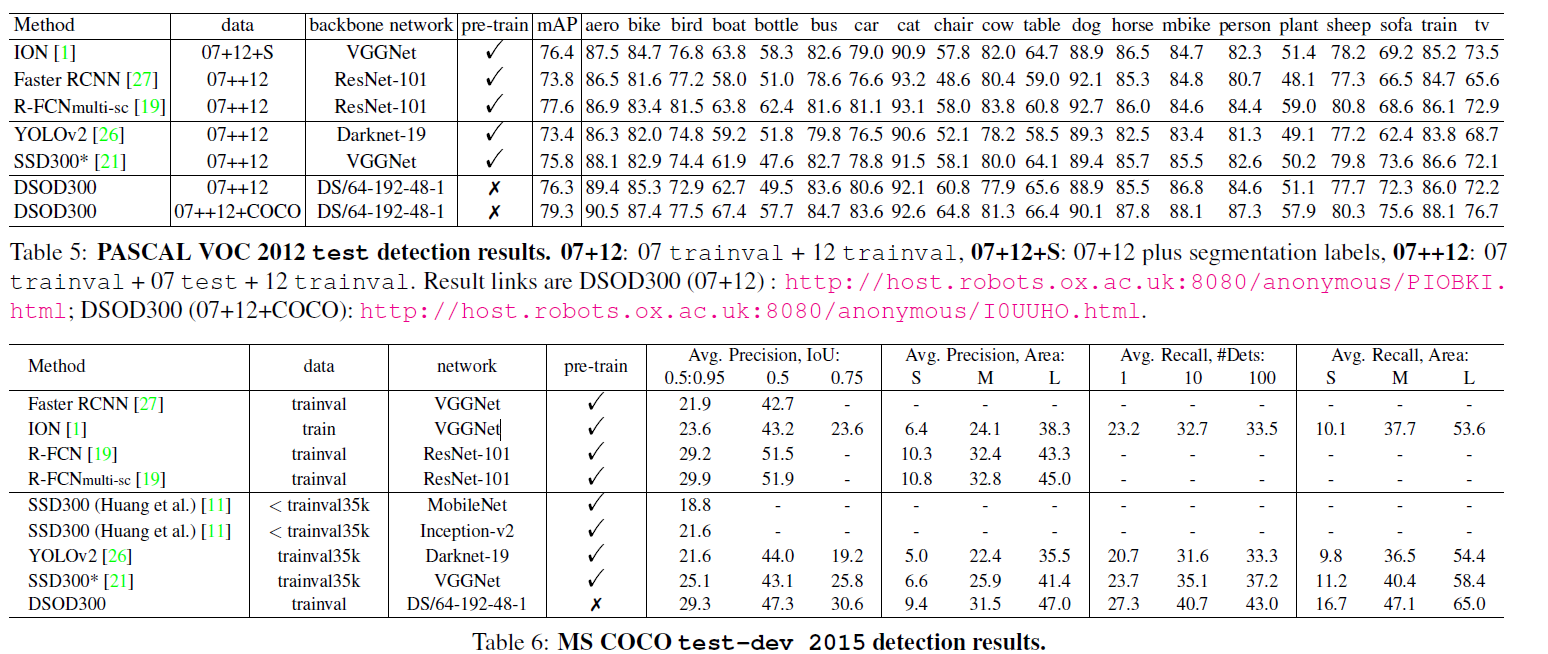
Learning Half and Reusing Half

In DSOD, in each scale (except scale-1), half of the feature maps are learned from the previous

scale with a series of conv-layers, while the remaining half feature maps are directly down-sampled from the contiguous high-resolution feature maps.

The pooling layeraims to match resolution to current size during concatenation.The 1X1 conv-layer is used to reduce the number of channels to 50%.

**=====>Result**



<=============================================================================>

**=====>Title**

Feature-Fused SSD: Fast Detection for Small Objects

**=====>Main Contributions / New Opinions**

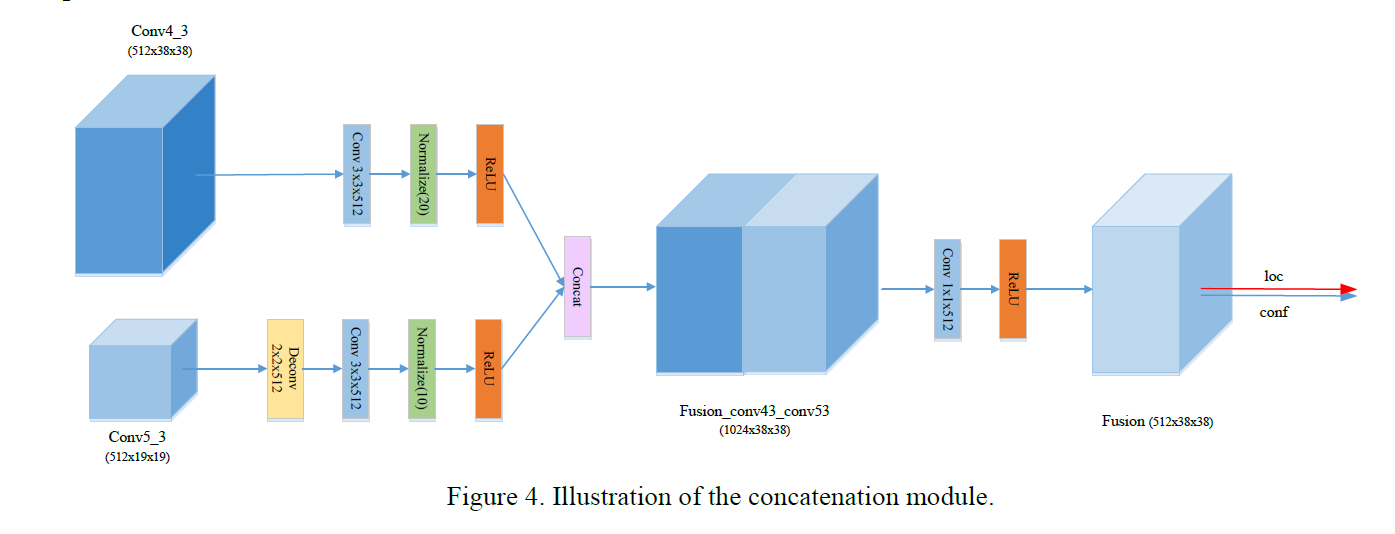
1.Feature Fusion

Conv4\_3、Conv5\_3 Concatenation

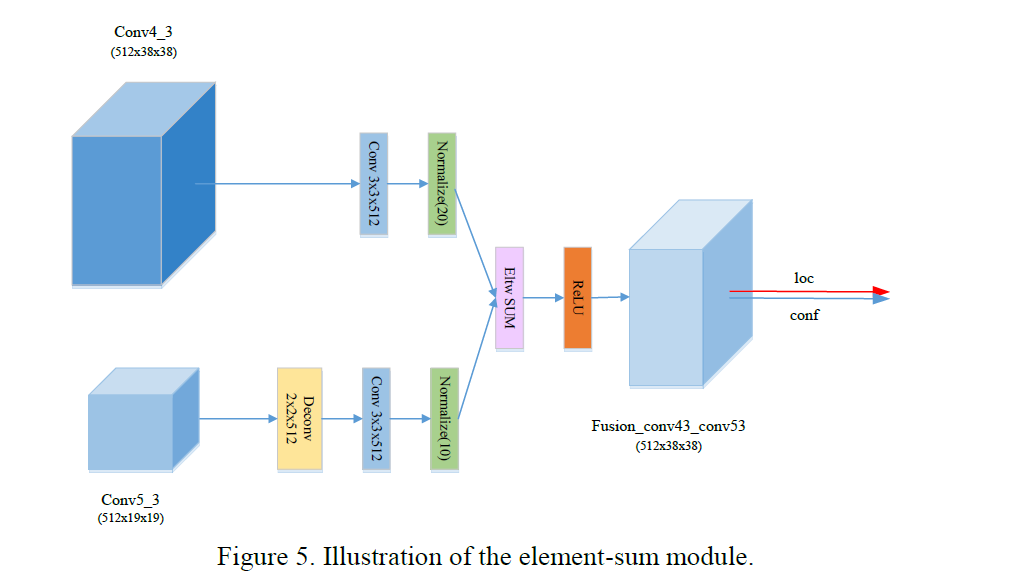
Conv4\_3、Conv5\_3 Element-wise sum

**=====>Key Points**

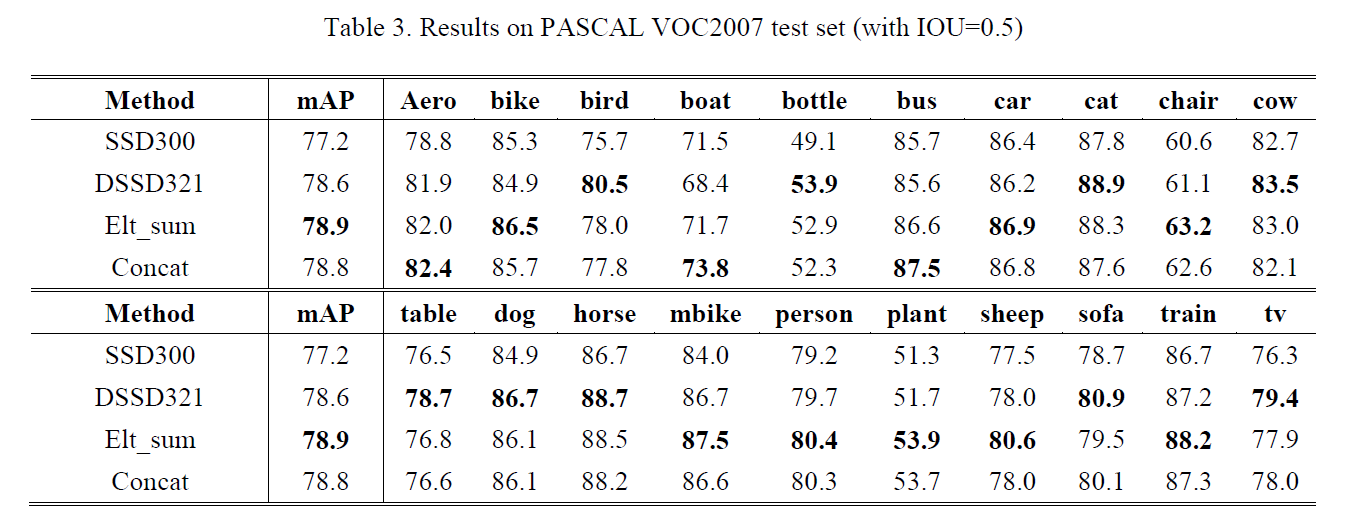
1.Concatenation



2.Element-wise sum



**=====>Result**



<=============================================================================>

**=====>Title**

LocNet: Improving Localization Accuracy for Object Detection

**=====>Background**

1.localization accuracy 少人问津

2.PASCAL VOC IOU=0.5 (object has been successfully detected)

3.Real life higher localization accuracy (e.g. IoU> 0.7) is normally required

4.COCO detection challenge 把IOU值也作为了最终的评价指标（MAP+IOU）

5.提高目标检测的IOU（而不仅是MAP）将会成为未来目标检测的主要挑战

6.传统的bbox回归：尝试直接通过回归的方式直接得到bbox的坐标，很难得到很准确的bbox

**=====>Main Contributions / New Opinions**

1.可以很方便的和现在最先进的目标检测系统结合

2.提出了两种基于行列的概率模型解决定位准确率，而不是回归的方式，并与回归方式进行了对比

3.对传统方法和最先进的方法不同iou下的map都有所提高

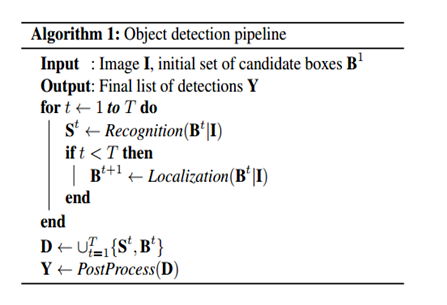
4.未来可以完全取代bbox回归的方法

**=====>Key Points**

1.两种概率模型

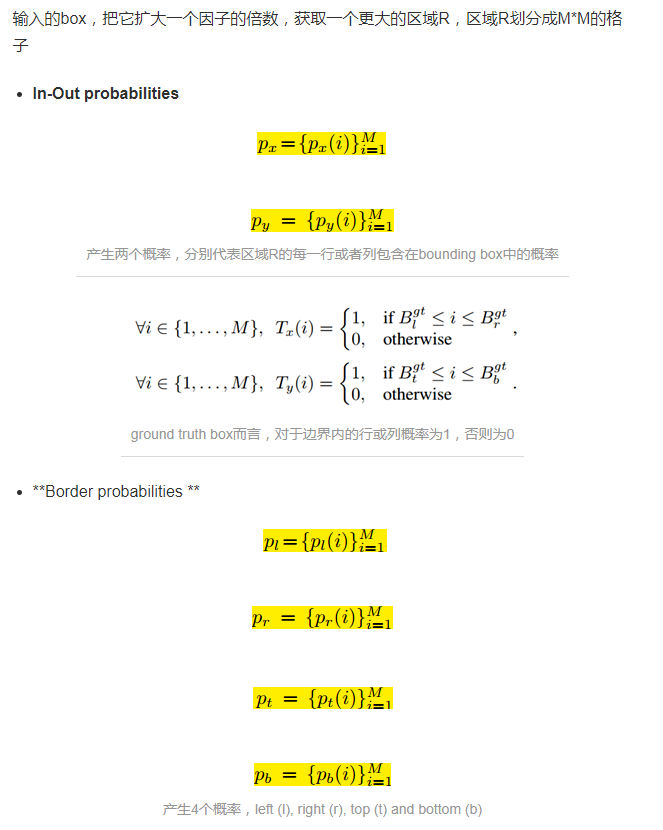


2.Detection Pipeline

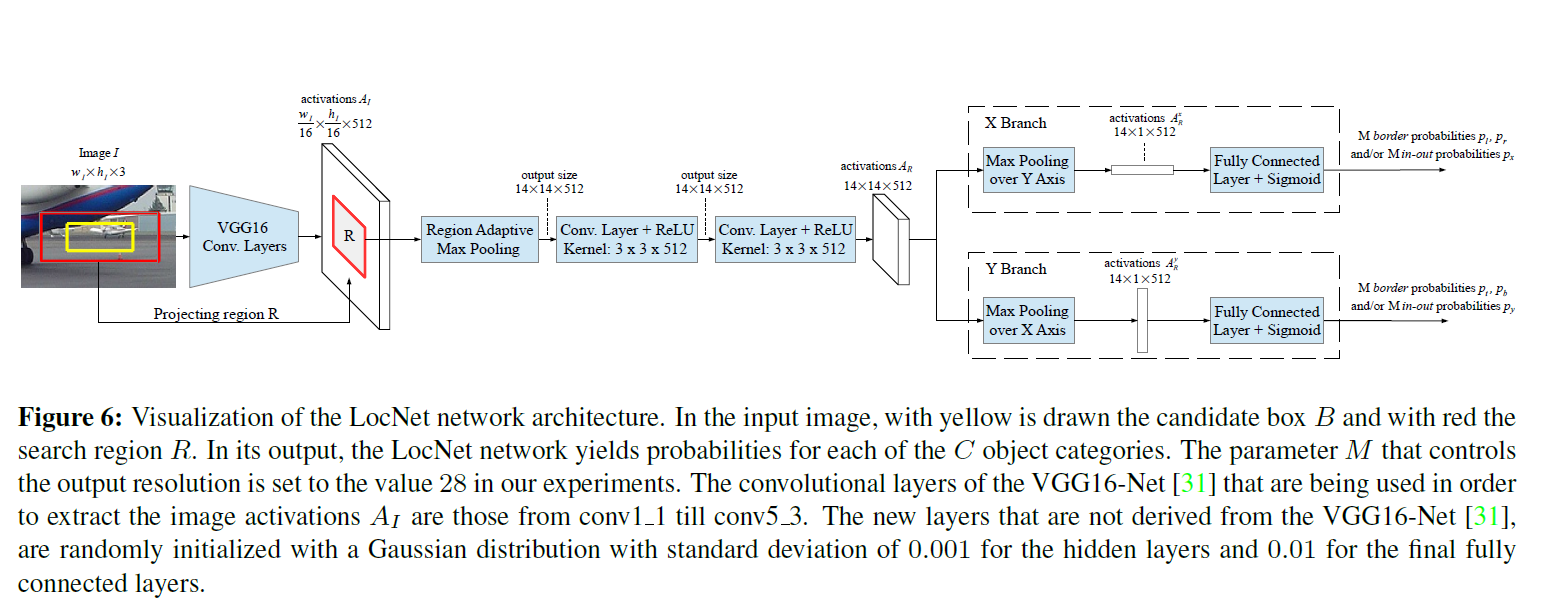


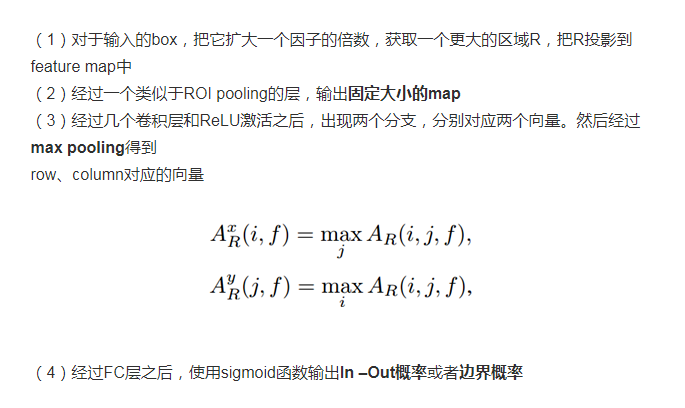


3.Model Prediction

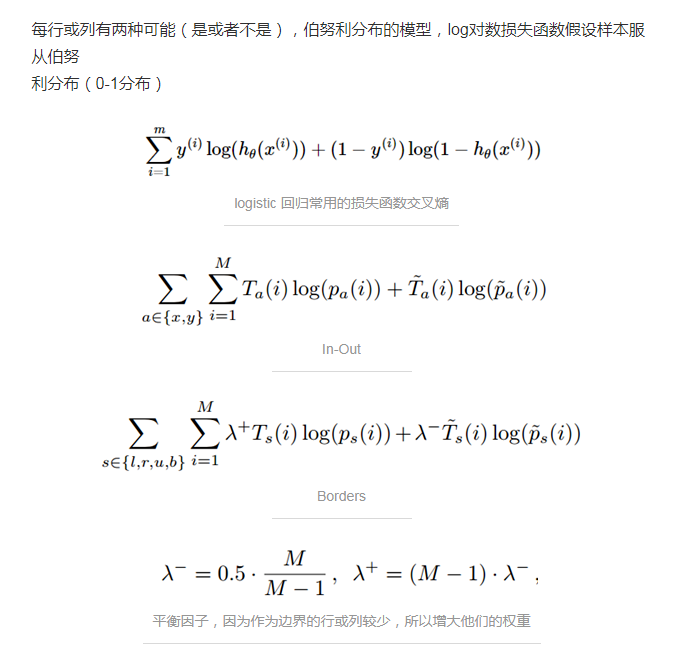


4.Network Architecture

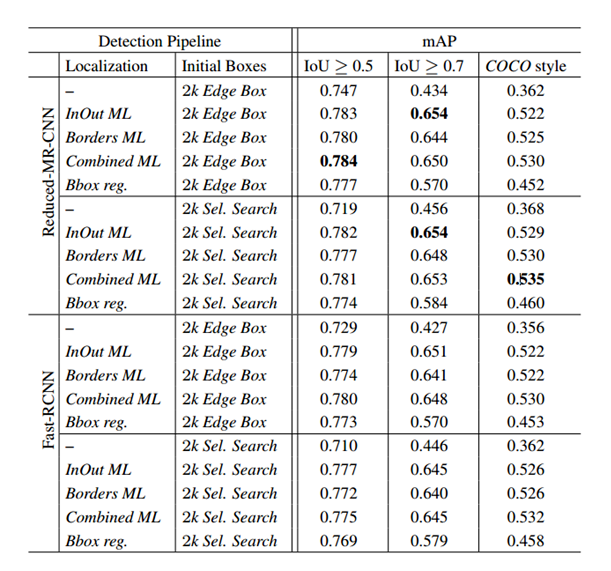


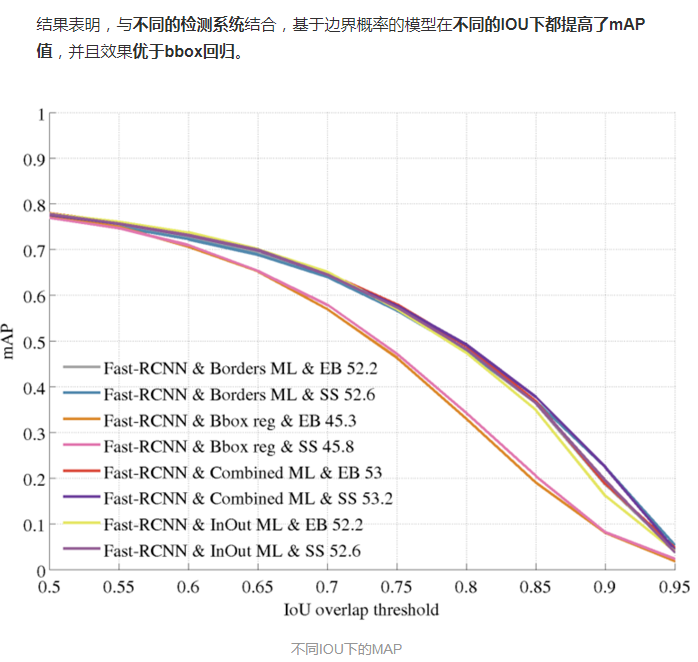


5.Loss Function



**=====>Result**





**=====>Title**

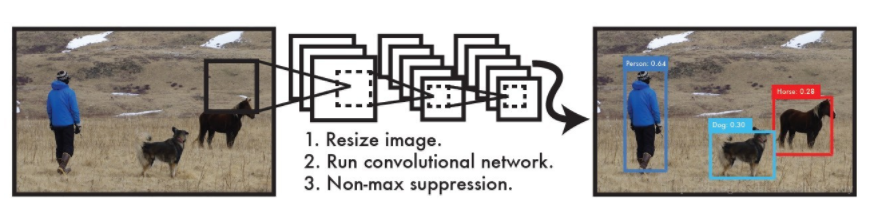
You Only Look Once: Unified, Real-Time Object Detection

**=====>Main Contributions / New Opinions**

RCNN、Fast RCNN、Faster RCNN系列方法通过region proposal产生大量的可能包含检测物体的potential bounding box，再用分类器去判断每个bounding box里面是否包含物体，以及物体所属类别的probability或者confidence。

YOLO将物体检测当作regression问题来处理，使用神经网络，直接从一张图像中预测出bounding box的坐标，box中包含物体的置信度和物体的probability。因为YOLO的物体检测流程是在一个NN（Neural Network）中完成的，所以可以end to end来优化物体检测性能。

**=====>Key Points**



如图所示，使用YOLO检测物体，其流程如下：

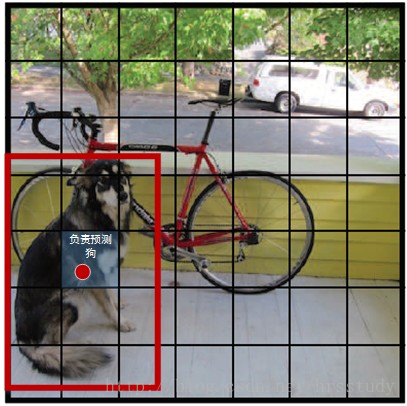
1.将图像resize到448\*448，作为输入

2.运行神经网络，得到bounding box坐标，box中包含物体的置信度和class probability

3.进行非极大值抑制，筛选boxes

Unified Detection

YOLO将输入图像划分为S\*S的栅格，每个栅格负责检测中心落在该栅格中的物体



每个栅格预测B个bounding boxes，以及这些bounding boxes的confidence scores。

该confidence scores反映了模型对于该栅格的预测：该栅格是否含有物体，以及这个box的坐标预测的有多准。

公式定义如下：

*Confidence=Pr(Object)\*IOU(truth, pred)*

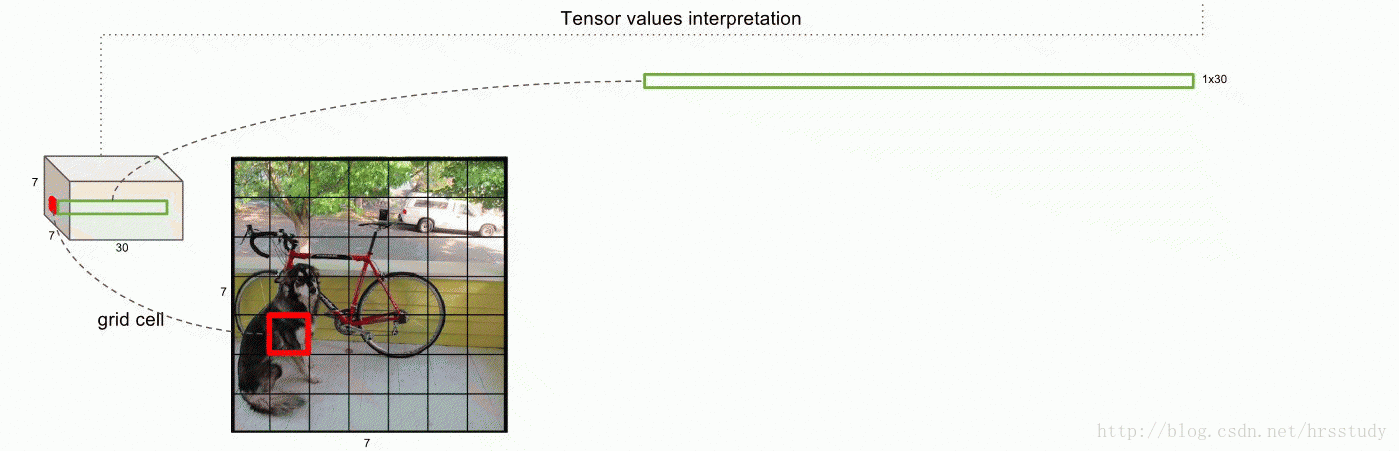
如果该栅格中不存在object时，则confidence score应该为0；否则，confidence score为predicted bounding box与ground truth box之间的IOU。

YOLO对每个bounding box有5个预测：x,y,w,h,confidence

坐标x，y代表了预测的bounding box的中心与栅格边界的相对值

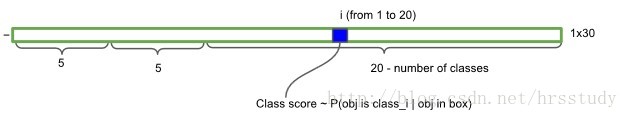
坐标w，h代表了预测的bounding box的width、height相对于整幅图像width，height的比例

Confidence就是预测的bounding box和ground truth box的IOU值



每个栅格除了预测B个bounding boxes，还要预测C个conditional class probability（条件类别概率）：Pr(Class|Object)。即在一个栅格包含一个object的前提下，它属于某个类别的概率。

我们只为每个栅格预测一组（C个）类概率，而不考虑框B的数量。



注意：

Conditional class probability信息是针对每个栅格的。

Confidence信息是针对每个bounding box的。

在测试阶段，将每个栅格的conditional class probability与每个bounding box的confidence相乘：

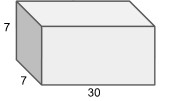
IMG_256

这样即可得到每个bounding box的具体类别的confidence score

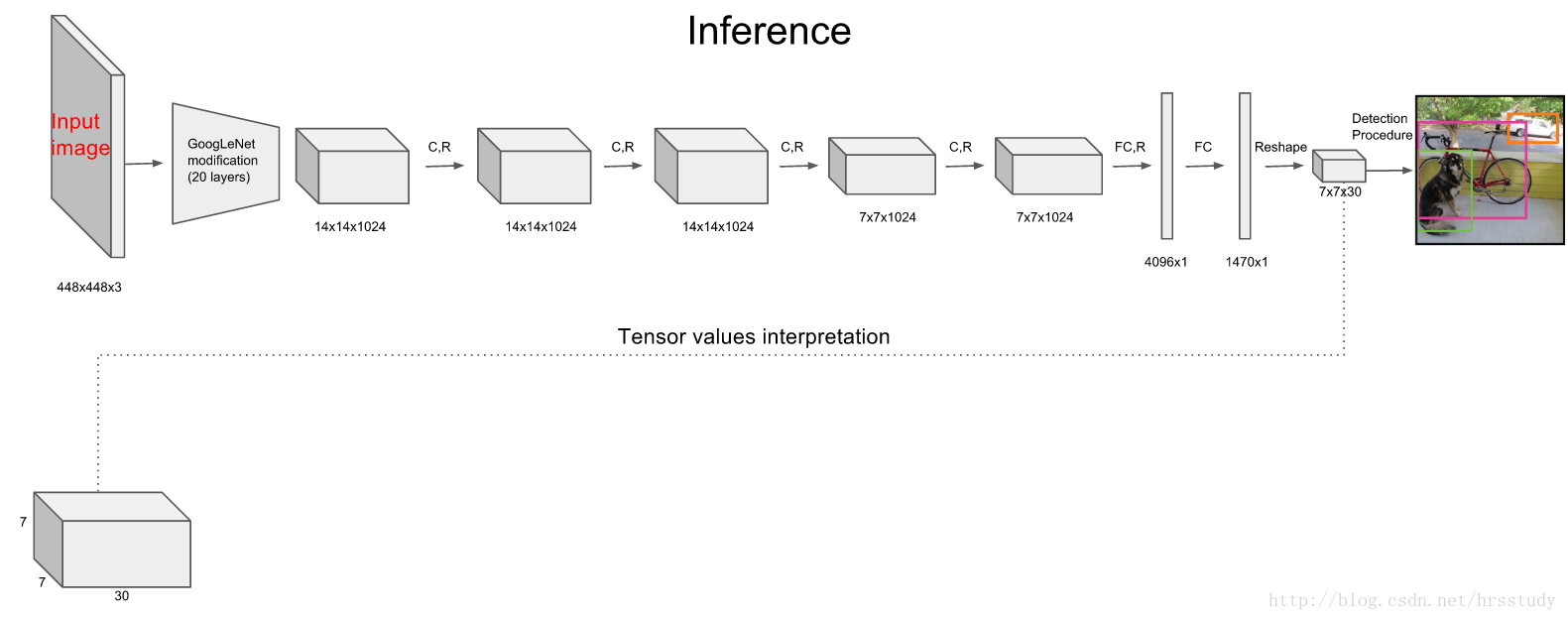
这乘积既包含了bounding box中预测的class的probability信息，也反映了bounding box是否含有Object和bounding box坐标的准确度。

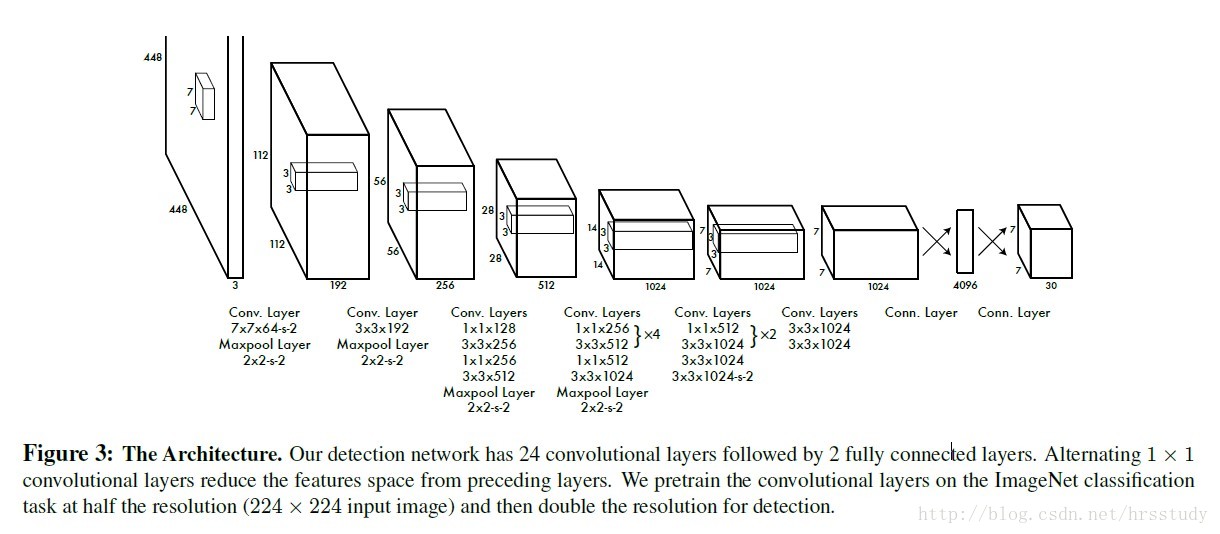
For Pascal VOC:

论文使用的S=7，即将一张图像分为7\*7=49个栅格，每个栅格预测B=2个boxes（每个box有x,y,w,h,confidence，5个预测值），同时C=20（Pascal VOC有20个类别），因此最终的prediction是7\*7\*30｛即S\*S\*(B\*5+C)｝的tensor。



Network Design





其中，卷积层用于提取图像特征，全连接层用于预测图像位置与类别概率

YOLO网络借鉴了GoogLeNet分类网络结构。不同的是，YOLO未使用inception module，而是使用1x1卷积层（此处1x1卷积层的存在是为了跨通道信息整合）+3x3卷积层简单替代。

Training

首先利用ImageNet 1000-class的分类任务数据集Pretrain卷积层。使用上述网络中的前20 个卷积层，加上一个 average-pooling layer，最后加一个全连接层，作为 Pretrain 的网络。训练大约一周的时间，使得在ImageNet 2012的验证数据集Top-5的精度达到 88%，这个结果跟 GoogleNet 的效果相当。

将Pretrain的结果的前20层卷积层应用到Detection中，并加入剩下的4个卷积层及2个全连接。

同时为了获取更精细化的结果，将输入图像的分辨率由 224\* 224 提升到 448\* 448。

将所有的预测结果都归一化到 0~1, 使用 Leaky RELU 作为激活函数。

为了防止过拟合，在第一个全连接层后面接了一个 ratio=0.5 的Dropout层。

为了提高精度，对原始图像做数据提升。

损失函数

损失函数的设计目标就是让坐标（x,y,w,h），confidence，classification 这个三个方面达到很好的平衡。

简单的全部采用了sum-squared error loss来做这件事会有以下不足：

a) 8维的localization error和20维的classification error同等重要显然是不合理的。

b) 如果一些栅格中没有object（一幅图中这种栅格很多），那么就会将这些栅格中的bounding box的confidence 置为0，相比于较少的有object的栅格，这些不包含物体的栅格对梯度更新的贡献会远大于包含物体的栅格对梯度更新的贡献，这会导致网络不稳定甚至发散。



解决方案如下：

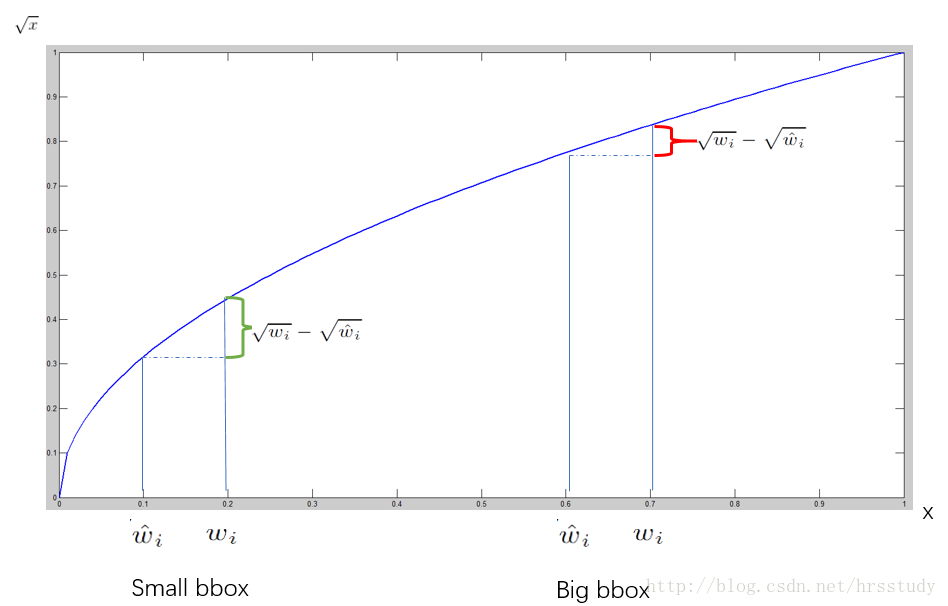
更重视8维的坐标预测，给这些损失前面赋予更大的loss weight, 记为 λcoord ,在pascal VOC训练中取5。（上图蓝色框）

对没有object的bbox的confidence loss，赋予小的loss weight，记为 λnoobj ，在pascal VOC训练中取0.5。（上图橙色框）

有object的bbox的confidence loss (上图红色框) 和类别的loss （上图紫色框）的loss weight正常取1。

对不同大小的bbox预测中，相比于大bbox预测偏一点，小box预测偏相同的尺寸对IOU的影响更大。而sum-square error loss中对同样的偏移loss是一样。

为了缓和这个问题，作者用了一个巧妙的办法，就是将box的width和height取平方根代替原本的height和width。 如下图：small bbox的横轴值较小，发生偏移时，反应到y轴上的loss（下图绿色）比big box(下图红色)要大

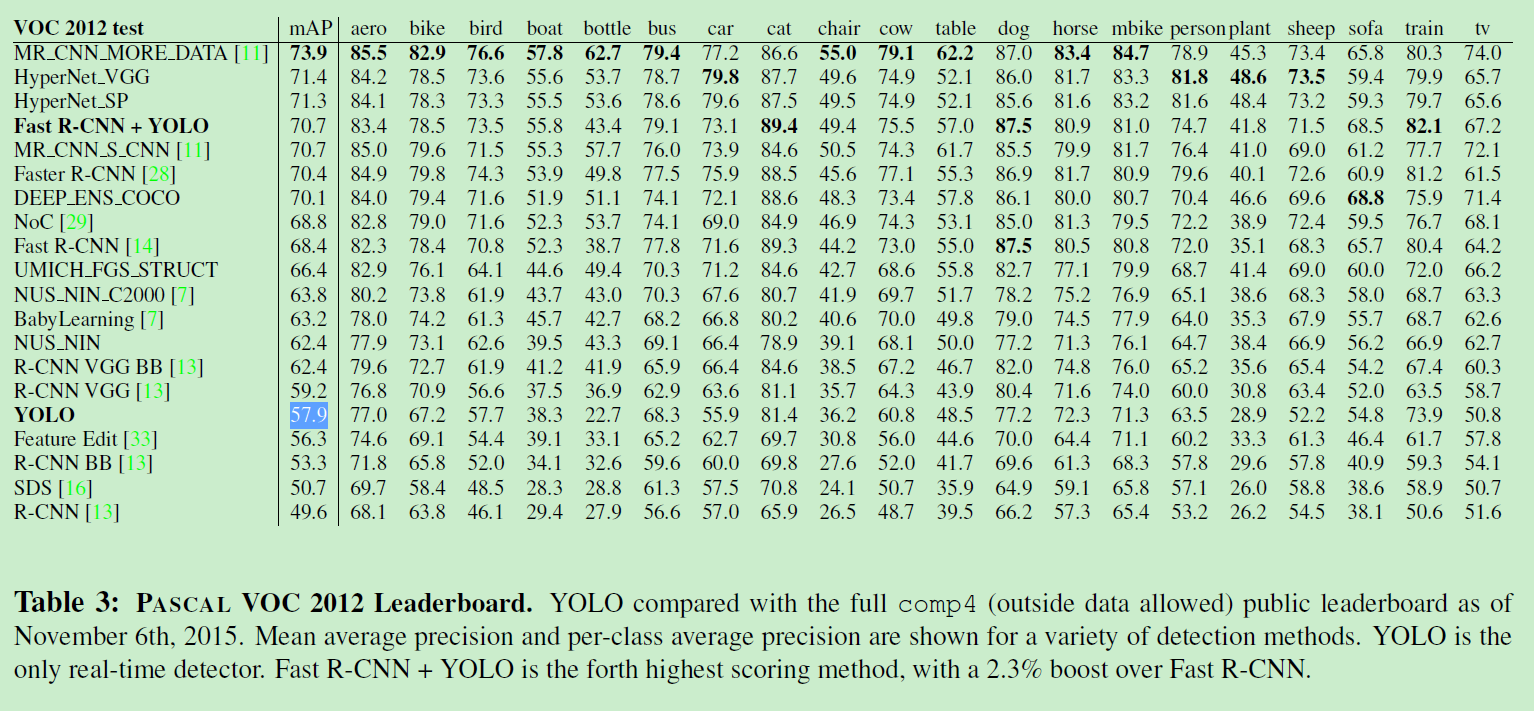


在 YOLO中，每个栅格预测多个bounding box，但在网络模型的训练中，希望每一个物体最后由一个bounding box predictor来负责预测。

因此，当前哪一个predictor预测的bounding box与ground truth box的IOU最大，这个 predictor就负责 predict object。

这会使得每个predictor可以专门的负责特定的物体检测。随着训练的进行，每一个 predictor对特定的物体尺寸、长宽比的物体的类别的预测会越来越好。

**=====>Result**



Reference：*https://blog.csdn.net/hrsstudy/article/details/70305791*

**=====>Title**

YOLO9000: Better, Faster, Stronger

**=====>Main Contributions / New Opinions**

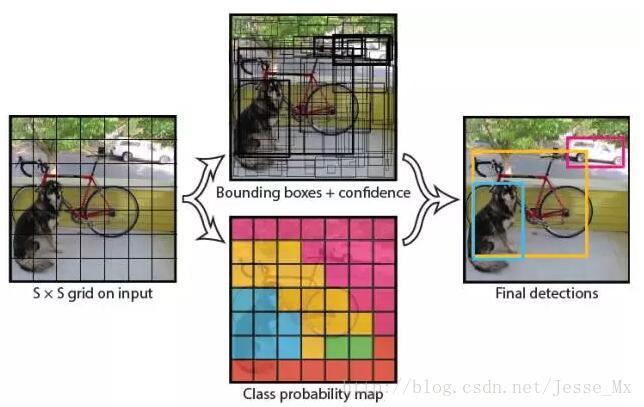
两方面的改进：

1.作者对原来的YOLO框架进行了改进，在保持原有速度的优势下，精度得以提升

2.作者提出了一种目标分类与检测的联合训练方法，通过该方法，YOLO9000可以同时在COCO和ImageNet数据集中同时进行训练，训练后的模型可以实现多达9000种物体的实时监测

**=====>Key Points**

YOLOv1



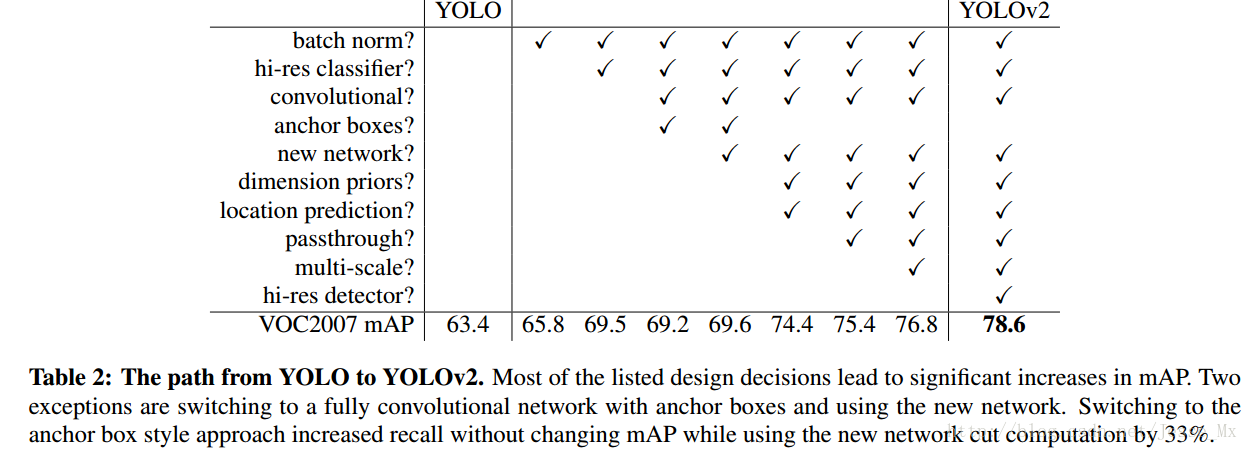
(1) 给个一个输入图像，首先将图像划分成7 \* 7的网格。

(2) 对于每个网格，每个网格预测2个bouding box（每个box包含5个预测量）以及20个类别概率，总共输出7×7×（2\*5+20）=1470个tensor

(3) 根据上一步可以预测出7 \* 7 \* 2 = 98个目标窗口，然后根据阈值去除可能性比较低的目标窗口，再由NMS去除冗余窗口即可。

YOLOv1使用了end-to-end的回归方法，没有region proposal步骤，直接回归便完成了位置和类别的判定。种种原因使得YOLOv1在目标定位上不那么精准，直接导致YOLO的检测精度并不是很高。

YOLOv2改进



Batch Normalization

CNN在训练过程中网络每层输入的分布一直在改变, 会使训练过程难度加大，但可以通过normalize每层的输入解决这个问题。新的YOLO网络在每一个卷积层后添加batch normalization，通过这一方法，mAP获得了2%的提升。batch normalization 也有助于规范化模型，可以在舍弃dropout优化后依然不会过拟合。

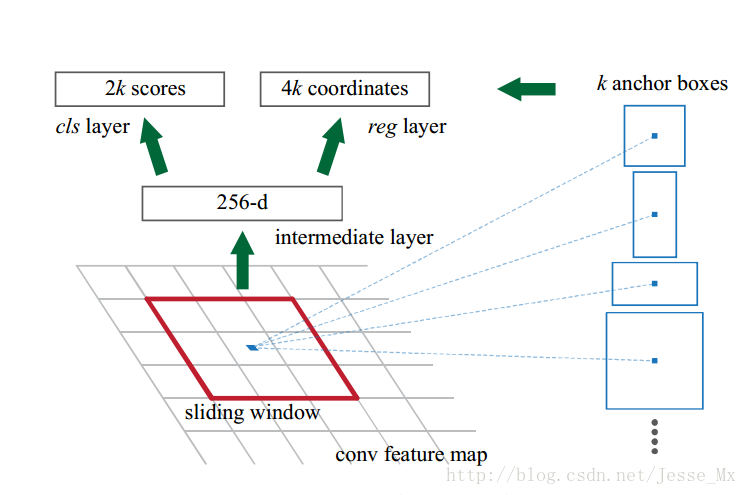
High Resolution Classifier

目前的目标检测方法中，基本上都会使用ImageNet预训练过的模型（classifier）来提取特征，如果用的是AlexNet网络，那么输入图片会被resize到不足256 \* 256，导致分辨率不够高，给检测带来困难。为此，新的YOLO网络把分辨率直接提升到了448 \* 448，这也意味之原有的网络模型必须进行某种调整以适应新的分辨率输入。

对于YOLOv2，作者首先对分类网络（自定义的darknet）进行了fine tune，分辨率改成448 \* 448，在ImageNet数据集上训练10轮（10 epochs），训练后的网络就可以适应高分辨率的输入了。然后，作者对检测网络部分（也就是后半部分）也进行fine tune。这样通过提升输入的分辨率，mAP获得了4%的提升。

Convolutional With Anchor Boxes

之前的YOLO利用全连接层的数据完成边框的预测，导致丢失较多的空间信息，定位不准。作者在这一版本中借鉴了Faster R-CNN中的anchor思想，回顾一下，anchor是RNP网络中的一个关键步骤，说的是在卷积特征图上进行滑窗操作，每一个中心可以预测9种不同大小的建议框。看到YOLOv2的这一借鉴，我只能说SSD的作者是有先见之明的。



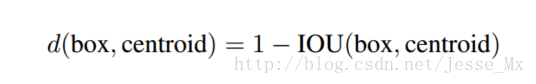
为了引入anchor boxes来预测bounding boxes，作者在网络中果断去掉了全连接层。剩下的具体怎么操作呢？首先，作者去掉了后面的一个池化层以确保输出的卷积特征图有更高的分辨率。然后，通过缩减网络，让图片输入分辨率为416 \* 416，这一步的目的是为了让后面产生的卷积特征图宽高都为奇数，这样就可以产生一个center cell。作者观察到，大物体通常占据了图像的中间位置， 就可以只用中心的一个cell来预测这些物体的位置，否则就要用中间的4个cell来进行预测，这个技巧可稍稍提升效率。最后，YOLOv2使用了卷积层降采样（factor为32），使得输入卷积网络的416 \* 416图片最终得到13 \* 13的卷积特征图（416/32=13）。

加入了anchor boxes后，可以预料到的结果是召回率上升，准确率下降。我们来计算一下，假设每个cell预测9个建议框，那么总共会预测13 \* 13 \* 9 = 1521个boxes，而之前的网络仅仅预测7 \* 7 \* 2 = 98个boxes。具体数据为：没有anchor boxes，模型recall为81%，mAP为69.5%；加入anchor boxes，模型recall为88%，mAP为69.2%。这样看来，准确率只有小幅度的下降，而召回率则提升了7%，说明可以通过进一步的工作来加强准确率，的确有改进空间。

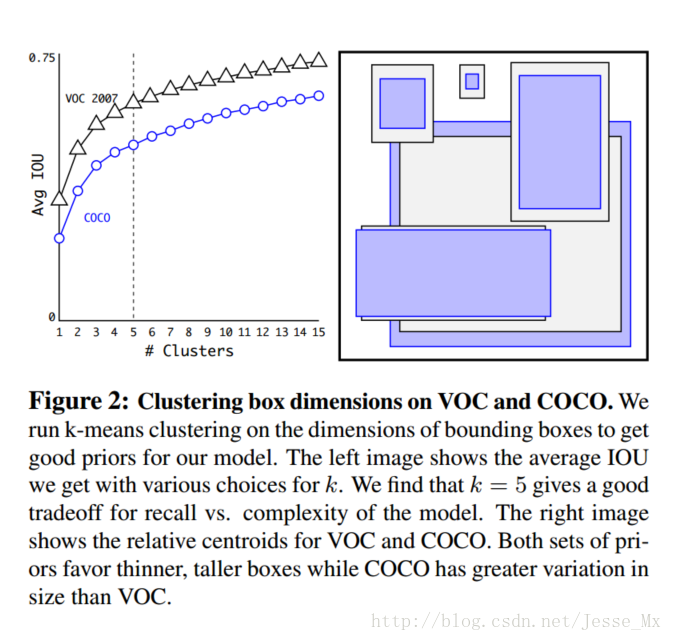
Dimension Clusters（维度聚类）

作者在使用anchor的时候遇到了两个问题，第一个是anchor boxes的宽高维度往往是精选的先验框（hand-picked priors），虽说在训练过程中网络也会学习调整boxes的宽高维度，最终得到准确的bounding boxes。但是，如果一开始就选择了更好的、更有代表性的先验boxes维度，那么网络就更容易学到准确的预测位置。

和以前的精选boxes维度不同，作者使用了K-means聚类方法类训练bounding boxes，可以自动找到更好的boxes宽高维度。传统的K-means聚类方法使用的是欧氏距离函数，也就意味着较大的boxes会比较小的boxes产生更多的error，聚类结果可能会偏离。为此，作者采用的评判标准是IOU得分（也就是boxes之间的交集除以并集），这样的话，error就和box的尺度无关了，最终的距离函数为：

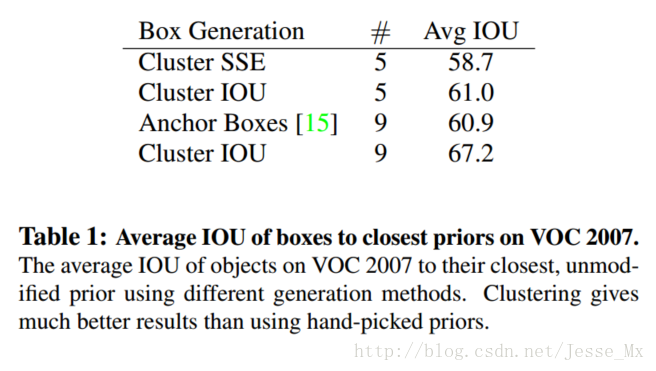


作者通过改进的K-means对训练集中的boxes进行了聚类，判别标准是平均IOU得分，聚类结果如下图：



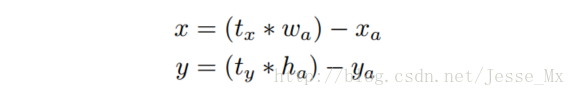
可以看到，平衡复杂度和IOU之后，最终得到k值为5，意味着作者选择了5种大小的box维度来进行定位预测，这与手动精选的box维度不同。结果中扁长的框较少，而瘦高的框更多（这符合行人的特征），这种结论如不通过聚类实验恐怕是发现不了的。

当然，作者也做了实验来对比两种策略的优劣，如下图，使用聚类方法，仅仅5种boxes的召回率就和Faster R-CNN的9种相当。说明K-means方法的引入使得生成的boxes更具有代表性，为后面的检测任务提供了便利。

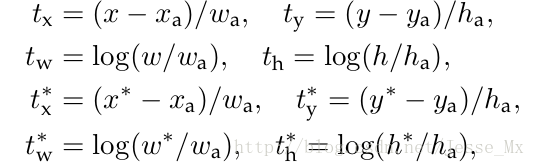


Direct location prediction（直接位置预测）

那么，作者在使用anchor boxes时发现的第二个问题就是：模型不稳定，尤其是在早期迭代的时候。大部分的不稳定现象出现在预测box的 (x,y) 坐标上了。在区域建议网络中，预测 (x,y) 以及 tx，ty 使用的是如下公式：



后来修改博文时，发现这个公式有误，作者应该是把加号写成了减号。理由如下，anchor的预测公式来自于Faster-RCNN，我们来看看人家是怎么写的：



公式中，符号的含义解释一下：x 是坐标预测值，xa 是anchor坐标（预设固定值），x∗是坐标真实值（标注信息），其他变量 y，w，h 以此类推，t 变量是偏移量。然后把前两个公式变形，就可以得到正确的公式：

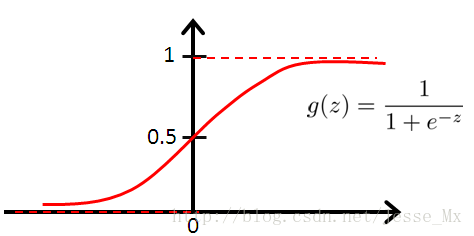
*x=(tx∗wa)+xa*

*y=(ty∗wa)+ya*

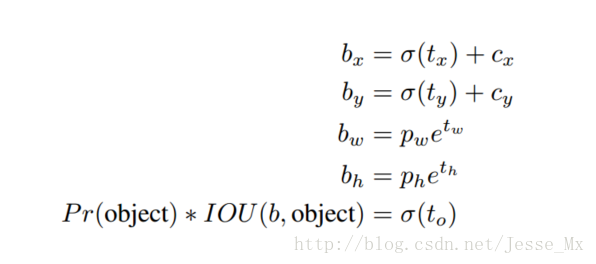
这个公式的理解为：当预测 tx=1，就会把box向右边移动一定距离（具体为anchor box的宽度），预测 tx=−1，就会把box向左边移动相同的距离。

这个公式没有任何限制，使得无论在什么位置进行预测，任何anchor boxes可以在图像中任意一点结束（我的理解是，tx 没有数值限定，可能会出现anchor检测很远的目标box的情况，效率比较低。正确做法应该是每一个anchor只负责检测周围正负一个单位以内的目标box）。模型随机初始化后，需要花很长一段时间才能稳定预测敏感的物体位置。

在此，作者就没有采用预测直接的offset的方法，而使用了预测相对于grid cell的坐标位置的办法，作者又把ground truth限制在了0到1之间，利用logistic回归函数来进行这一限制。



现在，神经网络在特征图（13 \*13 ）的每个cell上预测5个bounding boxes（聚类得出的值），同时每一个bounding box预测5个坐值，分别为 tx,ty,tw,th,to ，其中前四个是坐标，to是置信度。如果这个cell距离图像左上角的边距为 (cx,cy) 以及该cell对应box（bounding box prior）的长和宽分别为 (pw,ph)，那么预测值可以表示为：



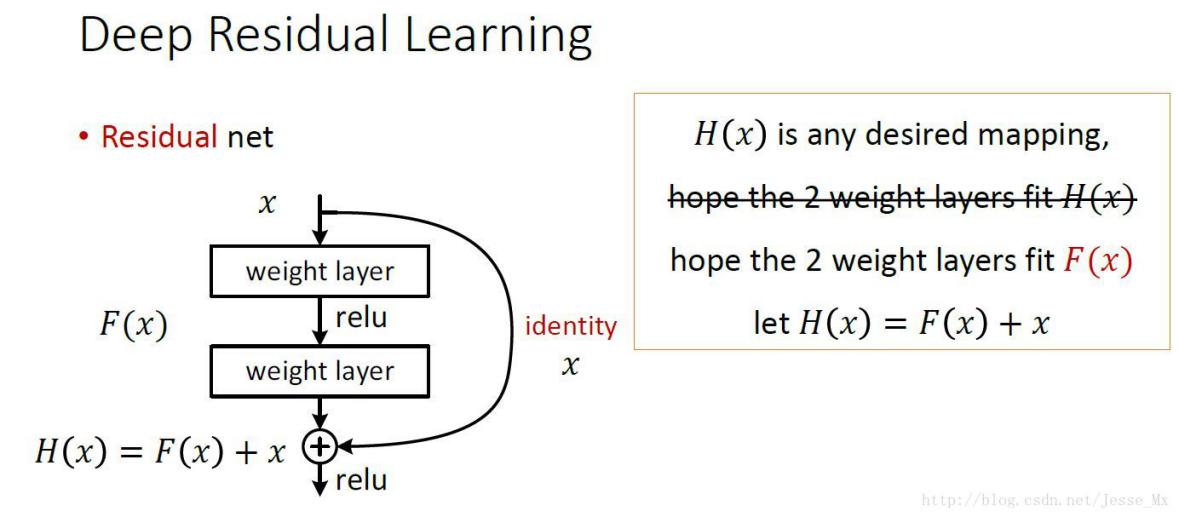
这几个公式参考上面Faster-RCNN和YOLOv1的公式以及下图就比较容易理解。tx,ty 经sigmod函数处理过，取值限定在了0~1，实际意义就是使anchor只负责周围的box，有利于提升效率和网络收敛。σ 函数的意义没有给，但估计是把归一化值转化为图中真实值，使用 e 的幂函数是因为前面做了 ln 计算，因此，σ(tx)是bounding box的中心相对栅格左上角的横坐标，σ(ty)是纵坐标，σ(to)是bounding box的confidence score。

定位预测值被归一化后，参数就更容易得到学习，模型就更稳定。作者使用Dimension Clusters和Direct location prediction这两项anchor boxes改进方法，mAP获得了5%的提升。



Fine-Grained Features（细粒度特征）

上述网络上的修改使YOLO最终在13 \* 13的特征图上进行预测，虽然这足以胜任大尺度物体的检测，但是用上细粒度特征的话，这可能对小尺度的物体检测有帮助。Faser R-CNN和SSD都在不同层次的特征图上产生区域建议（SSD直接就可看得出来这一点），获得了多尺度的适应性。这里使用了一种不同的方法，简单添加了一个转移层（ passthrough layer），这一层要把浅层特征图（分辨率为26 \* 26，是底层分辨率4倍）连接到深层特征图。



Multi-Scale Training

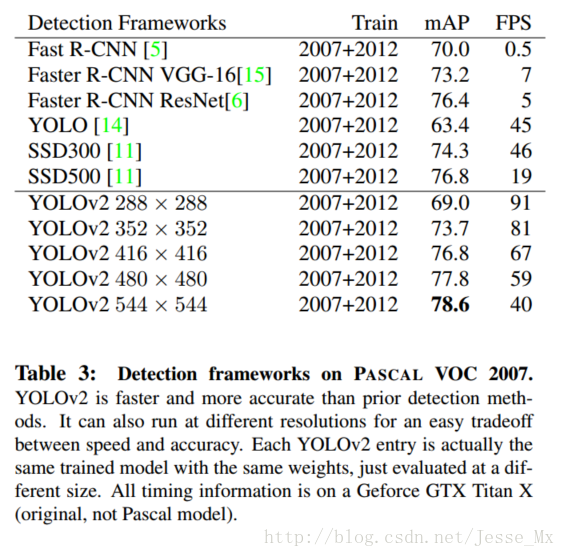
原来的YOLO网络使用固定的448 \* 448的图片作为输入，现在加入anchor boxes后，输入变成了416 \* 416。目前的网络只用到了卷积层和池化层，那么就可以进行动态调整（意思是可检测任意大小图片）。作者希望YOLOv2具有不同尺寸图片的鲁棒性，因此在训练的时候也考虑了这一点。

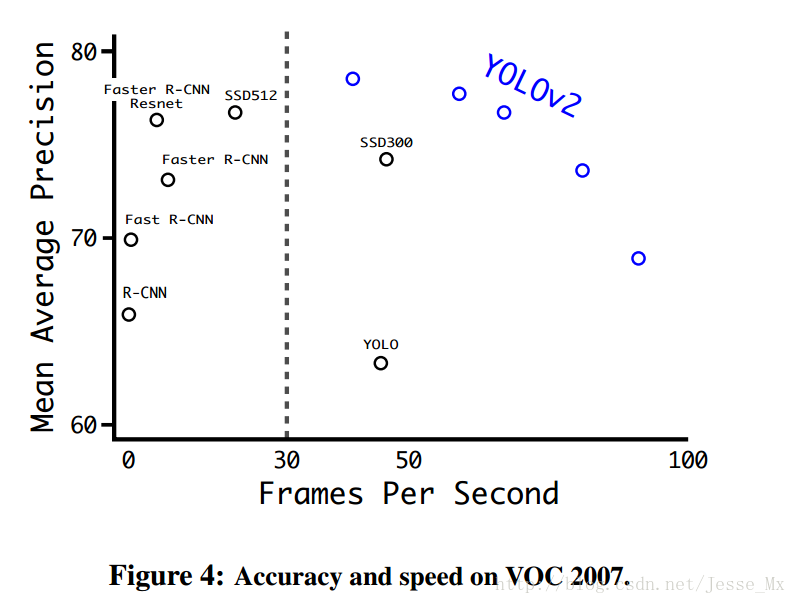
不同于固定输入网络的图片尺寸的方法，作者在几次迭代后就会微调网络。没经过10次训练（10 epoch），就会随机选择新的图片尺寸。YOLO网络使用的降采样参数为32，那么就使用32的倍数进行尺度池化{320,352，…，608}。最终最小的尺寸为320 \* 320，最大的尺寸为608 \* 608。接着按照输入尺寸调整网络进行训练。

这种机制使得网络可以更好地预测不同尺寸的图片，意味着同一个网络可以进行不同分辨率的检测任务，在小尺寸图片上YOLOv2运行更快，在速度和精度上达到了平衡。

在小尺寸图片检测中，YOLOv2成绩很好，输入为228 \* 228的时候，帧率达到90FPS，mAP几乎和Faster R-CNN的水准相同。使得其在低性能GPU、高帧率视频、多路视频场景中更加适用。

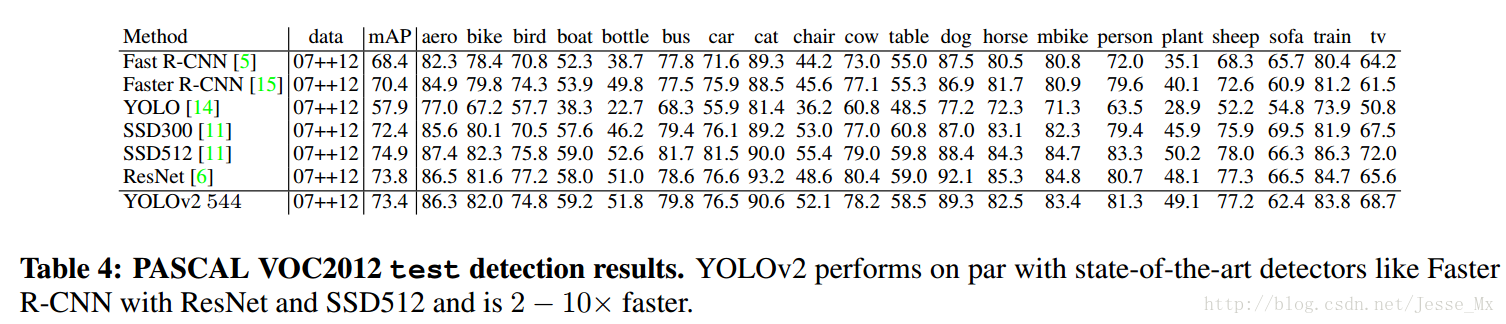
在大尺寸图片检测中，YOLOv2达到了先进水平，VOC2007 上mAP为78.6%，仍然高于平均水准，下图是YOLOv2和其他网络的成绩对比：

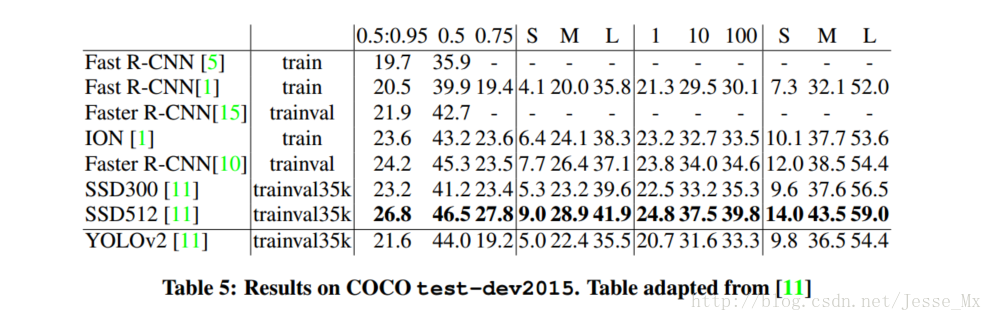




Further Experiments

作者在VOC2012上对YOLOv2进行训练，下图是和其他方法的对比。YOLOv2精度达到了73.4%，并且速度更快。同时YOLOV2也在COCO上做了测试（IOU=0.5），也和Faster R-CNN、SSD作了成绩对比。总的来说，比上不足，比下有余。





速度的改进（Faster）

YOLO一向是速度和精度并重，作者为了改善检测速度，也作了一些相关工作。

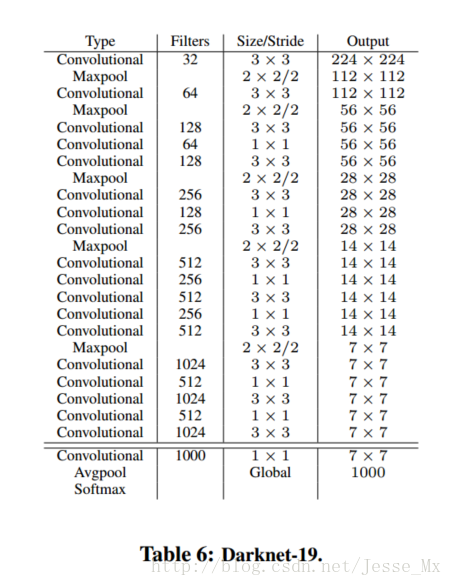
大多数检测网络有赖于VGG-16作为特征提取部分，VGG-16的确是一个强大而准确的分类网络，但是复杂度有些冗余。224 \* 224的图片进行一次前向传播，其卷积层就需要多达306.9亿次浮点数运算。

YOLOv2使用的是基于Googlenet的定制网络，比VGG-16更快，一次前向传播仅需85.2亿次运算。可是它的精度要略低于VGG-16，单张224 \* 224取前五个预测概率的对比成绩为88%和90%（低一点点也是可以接受的）。

Darknet-19

YOLOv2使用了一个新的分类网络作为特征提取部分，参考了前人的先进经验，比如类似于VGG，作者使用了较多的3 \* 3卷积核，在每一次池化操作后把通道数翻倍。借鉴了network in network的思想，网络使用了全局平均池化（global average pooling），把1 \* 1的卷积核置于3 \* 3的卷积核之间，用来压缩特征。也用了batch normalization（前面介绍过）稳定模型训练。

最终得出的基础模型就是Darknet-19，如下图，其包含19个卷积层、5个最大值池化层（maxpooling layers ），下图展示网络具体结构。Darknet-19运算次数为55.8亿次，imagenet图片分类top-1准确率72.9%，top-5准确率91.2%。



Training for classification

作者使用Darknet-19在标准1000类的ImageNet上训练了160次，用的随机梯度下降法，starting learning rate 为0.1，polynomial rate decay 为4，weight decay为0.0005 ，momentum 为0.9。训练的时候仍然使用了很多常见的数据扩充方法（data augmentation），包括random crops, rotations, and hue, saturation, and exposure shifts。 （这些训练参数是基于darknet框架，和caffe不尽相同）

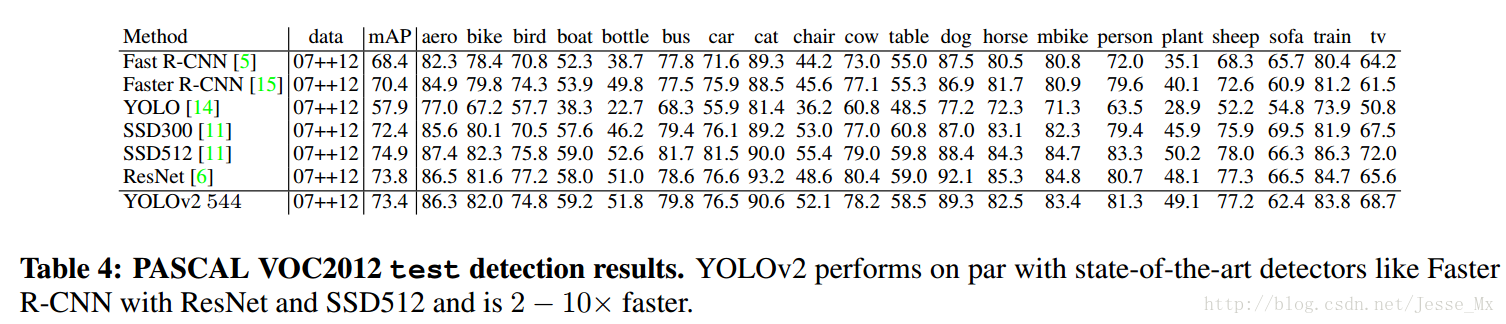
初始的224 \* 224训练后，作者把分辨率上调到了448 \* 448，然后又训练了10次，学习率调整到了0.001。高分辨率下训练的分类网络在top-1准确率76.5%，top-5准确率93.3%。

Training for detection

分类网络训练完后，就该训练检测网络了，作者去掉了原网络最后一个卷积层，转而增加了三个3 \* 3 \* 1024的卷积层（可参考darknet中cfg文件），并且在每一个上述卷积层后面跟一个1 \* 1的卷积层，输出维度是检测所需的数量。对于VOC数据集，预测5种boxes大小，每个box包含5个坐标值和20个类别，所以总共是5 \* （5+20）= 125个输出维度。同时也添加了转移层（passthrough layer ），从最后那个3 \* 3 \* 512的卷积层连到倒数第二层，使模型有了细粒度特征。

作者的检测模型以0.001的初始学习率训练了160次，在60次和90次的时候，学习率减为原来的十分之一。其他的方面，weight decay为0.0005，momentum为0.9，依然使用了类似于Faster-RCNN和SSD的数据扩充（data augmentation）策略。

**=====>Result**



**=====>Title**

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

**=====>Main Contributions / New Opinions**

解决region proposal耗时的问题。RCNN、Fast-RCNN采用SS(Selective Search)进行region proposal，该操作在CPU上进行，因此比较耗时。Faster RCNN提出了RPN(Region Proposal Network)用于区域提取，直接利用CNN，减少了区域提取的时间。

**=====>Key Points**

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。

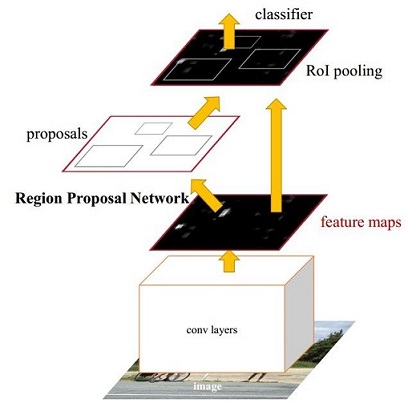


图1 Faster CNN基本结构（来自原论文）

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

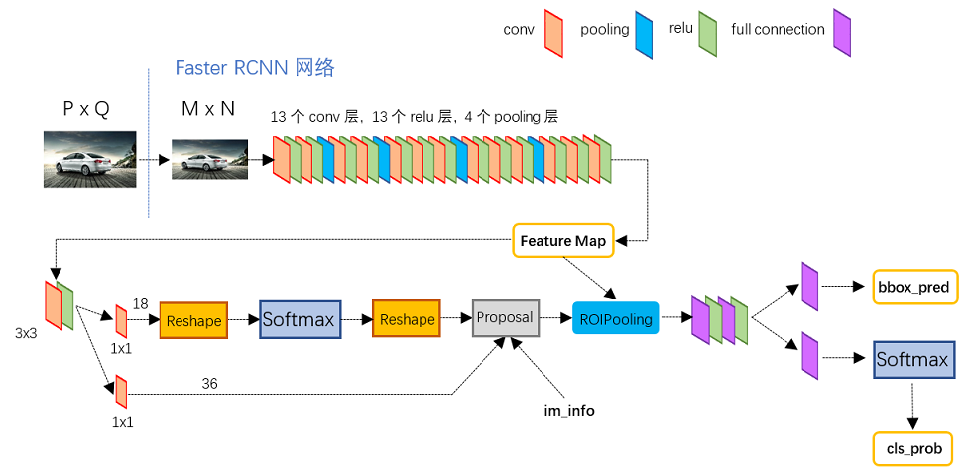


图2 faster\_rcnn\_test.pt网络结构

1 Conv layers

缩进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：

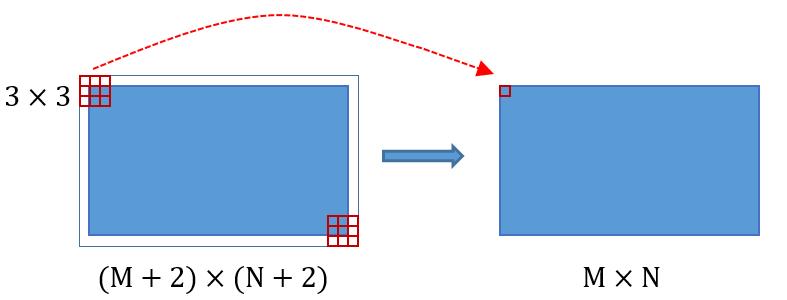


图3

2 Region Proposal Networks(RPN)

缩进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。

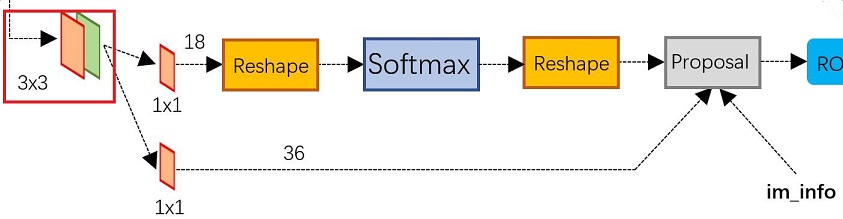


图4 RPN网络结构

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

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。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：

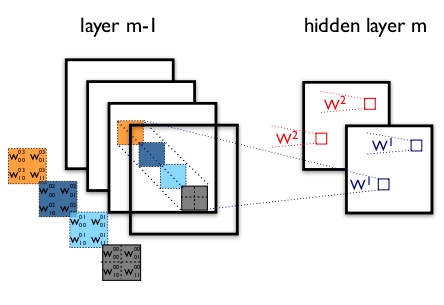


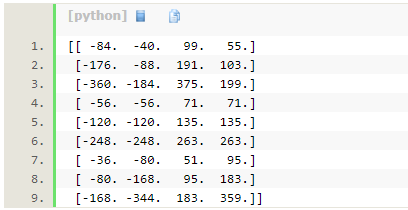
图5 多通道+多卷积核做卷积示意图（摘自Theano教程）

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！

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

缩进提到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：



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。

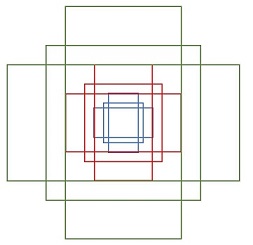


图6 anchors示意图

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。

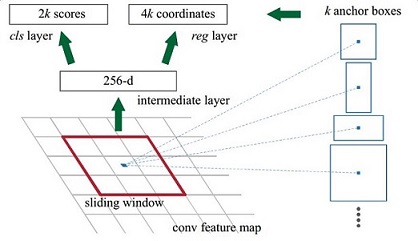


图7

解释一下上面这张图的数字。

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

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

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

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：

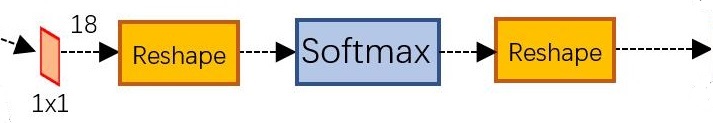
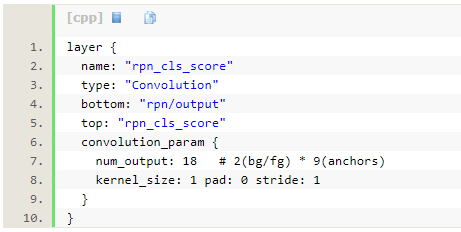


图8 RPN中判定fg/bg网络结构

该1x1卷积的caffe prototxt定义如下：

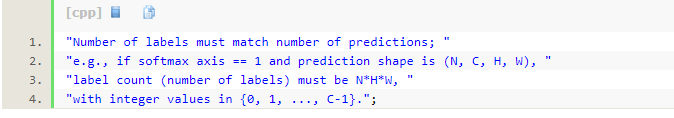


可以看到其num\_output=18，也就是经过该卷积的输出图像为WxHx18大小（注意第二章开头提到的卷积计算方式）。这也就刚好对应了feature maps每一个点都有9个anchors，同时每个anchors又有可能是foreground和background，所有这些信息都保存WxHx(9x2)大小的矩阵。为何这样做？后面接softmax分类获得foreground anchors，也就相当于初步提取了检测目标候选区域box（一般认为目标在foreground anchors中）。

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：



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。

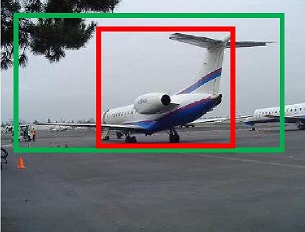


图9

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=(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)。

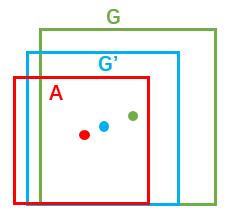


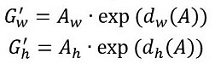
图10

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:

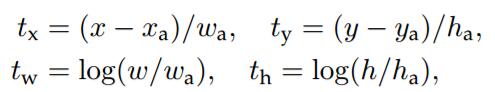
缩进 1. 先做平移

IMG_256

2. 再做缩放



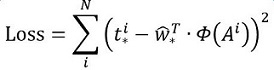
观察上面4个公式发现，需要学习的是dx(A)，dy(A)，dw(A)，dh(A)这四个变换。当输入的anchor A与GT相差较小时，可以认为这种变换是一种线性变换， 那么就可以用线性回归来建模对窗口进行微调（注意，只有当anchors A和GT比较接近时，才能使用线性回归模型，否则就是复杂的非线性问题了）。对应于Faster RCNN原文，平移量(tx, ty)与尺度因子(tw, th)如下：



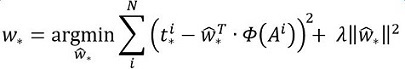
接下来的问题就是如何通过线性回归获得dx(A)，dy(A)，dw(A)，dh(A)了。线性回归就是给定输入的特征向量X, 学习一组参数W, 使得经过线性回归后的值跟真实值Y非常接近，即Y=WX。对于该问题，输入X是一张经过卷积获得的feature map，定义为Φ；同时还有训练传入的GT，即(tx, ty, tw, th)。输出是dx(A)，dy(A)，dw(A)，dh(A)四个变换。那么目标函数可以表示为：

IMG_256

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：



函数优化目标为：



2.5 对proposals进行bounding box regression

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

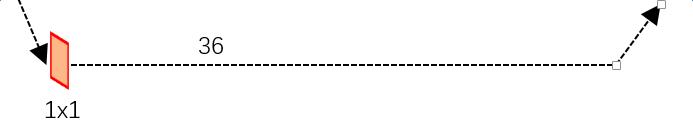
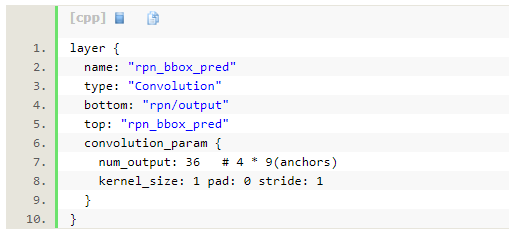


图11 RPN中的bbox reg

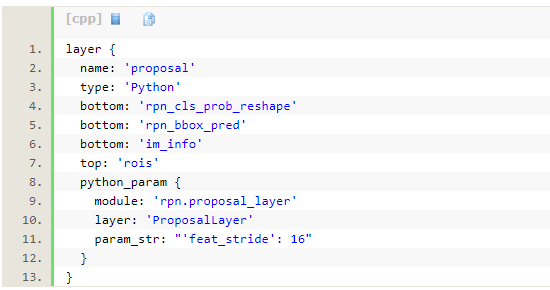
先来看一看上图11中1x1卷积的caffe prototxt定义：



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[dx(A)，dy(A)，dw(A)，dh(A)]变换量。

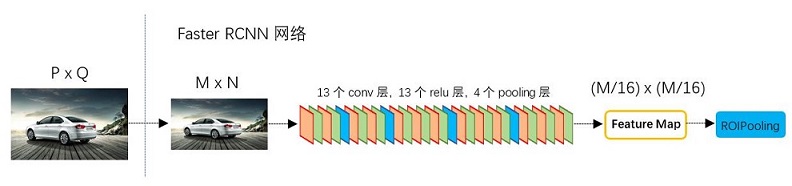
2.6 Proposal Layer

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。还是先来看看Proposal Layer的caffe prototxt定义：



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，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）

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

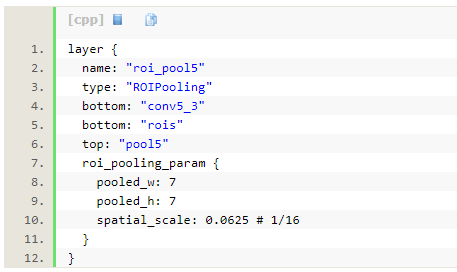


图13 crop与warp破坏图像原有结构信息

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

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



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。

缩进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）。

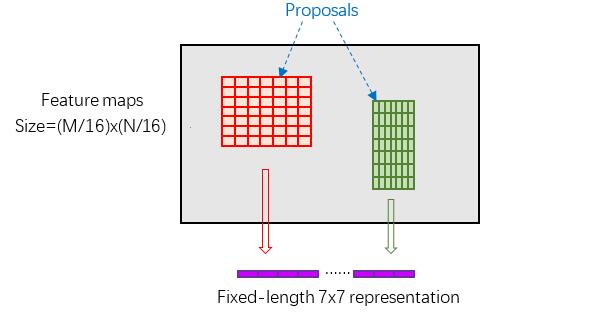


图14 proposal示意图

4 Classification

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。

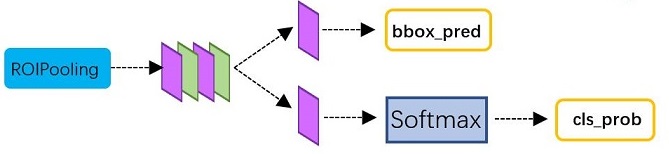


图15 Classification部分网络结构图

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

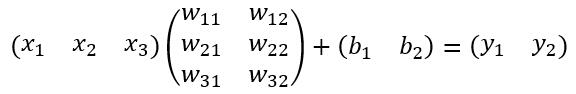
2.再次对proposals进行bounding box regression，获取更高精度的rect box

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：



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。

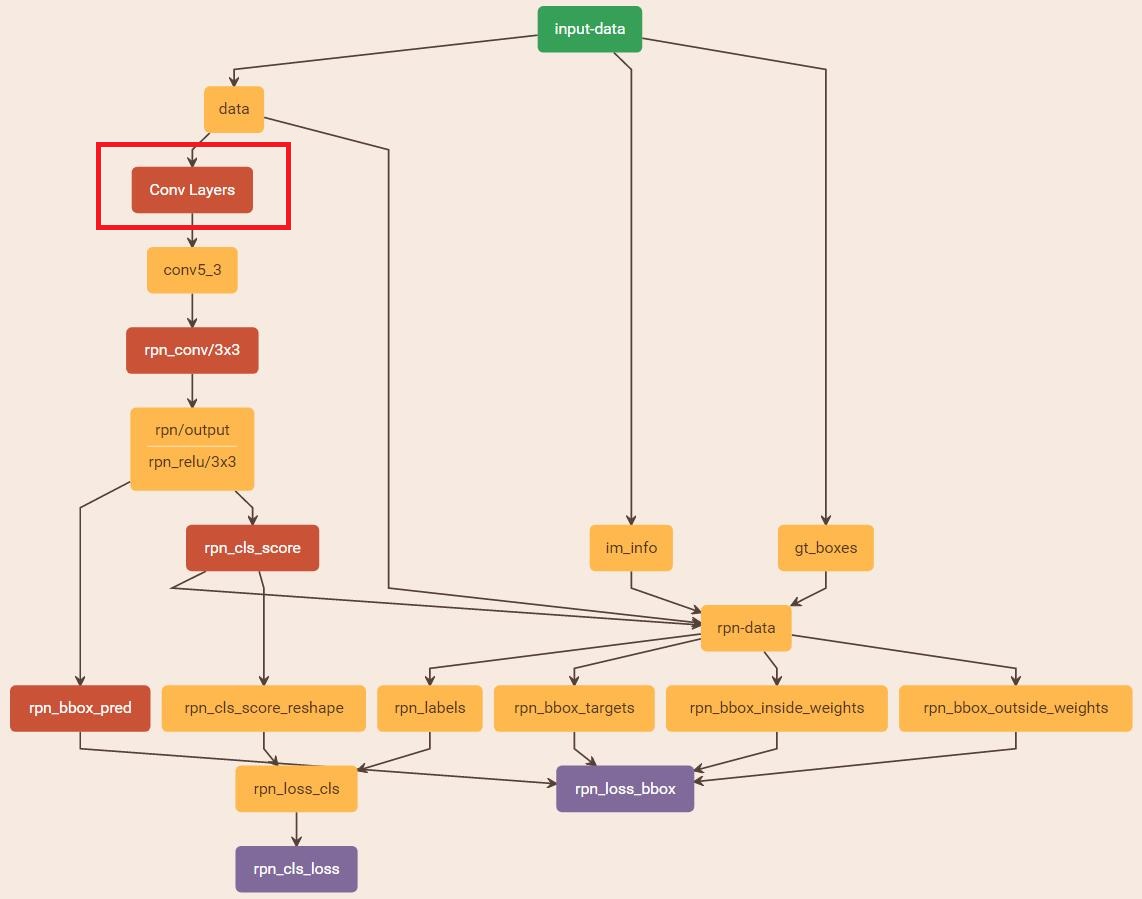
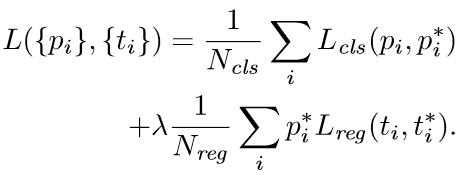


图17 stage1\_rpn\_train.pt

（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：

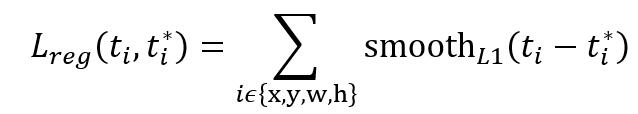


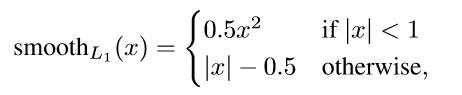
上述公式中，i表示anchors index，pi表示foreground softmax predict概率，pi\*代表对应的GT predict概率（即当第i个anchor与GT间IoU>0.7，认为是该anchor是foreground，pi\*=1；反之IoU<0.3时，认为是该anchor是background，pi\*=0；至于那些0.3<IoU<0.7的anchor则不参与训练）；t代表predict bounding box，t\*代表对应foreground anchor对应的GT box。可以看到，整个Loss分为2部分：

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

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）。

缩进由于在实际过程中，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2种Loss。这里比较重要是Lreg使用的soomth L1 loss，计算公式如下：





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\*，Ncls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，Nreg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

缩进在该步骤中，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。

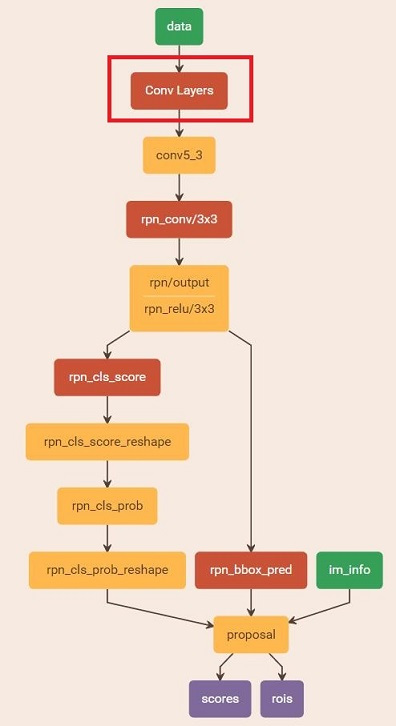


图18 rpn\_test.pt

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。

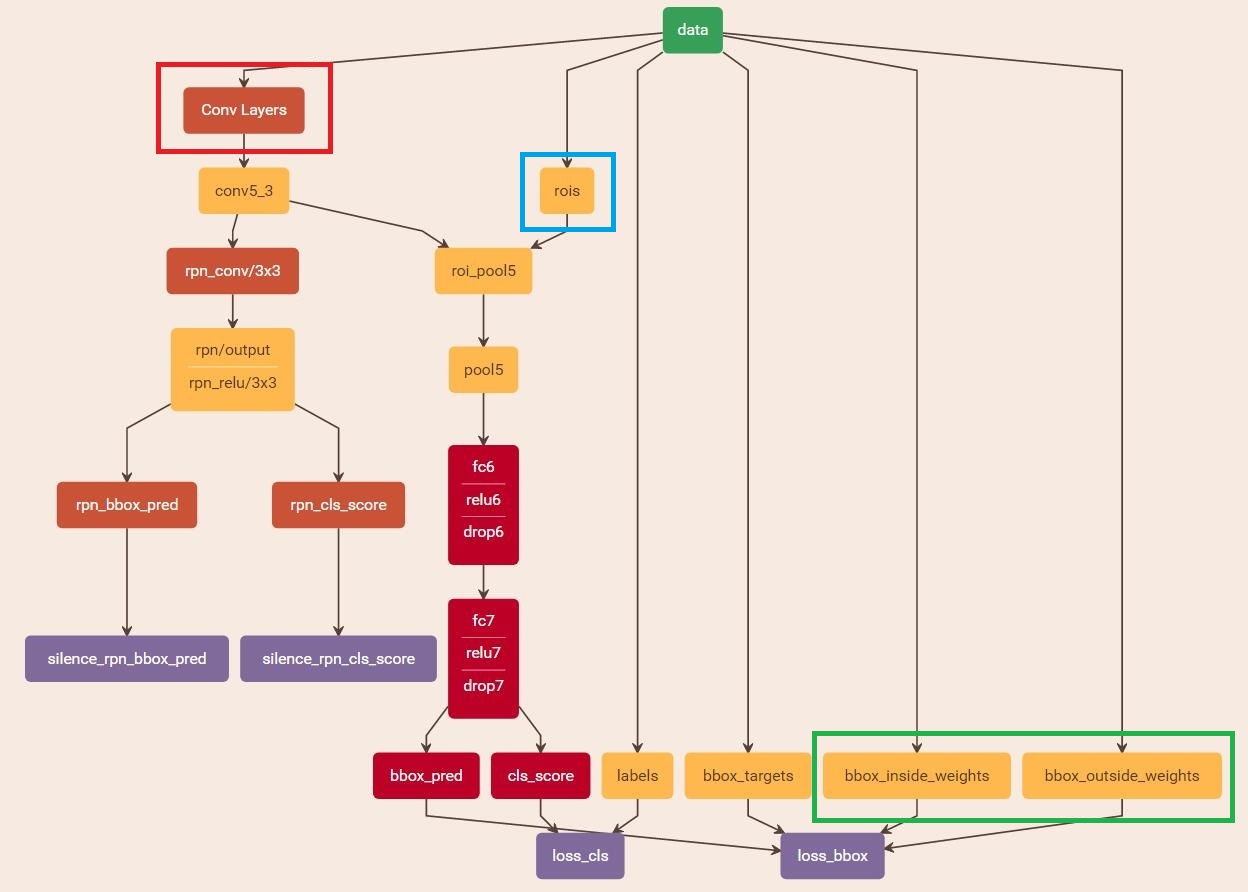
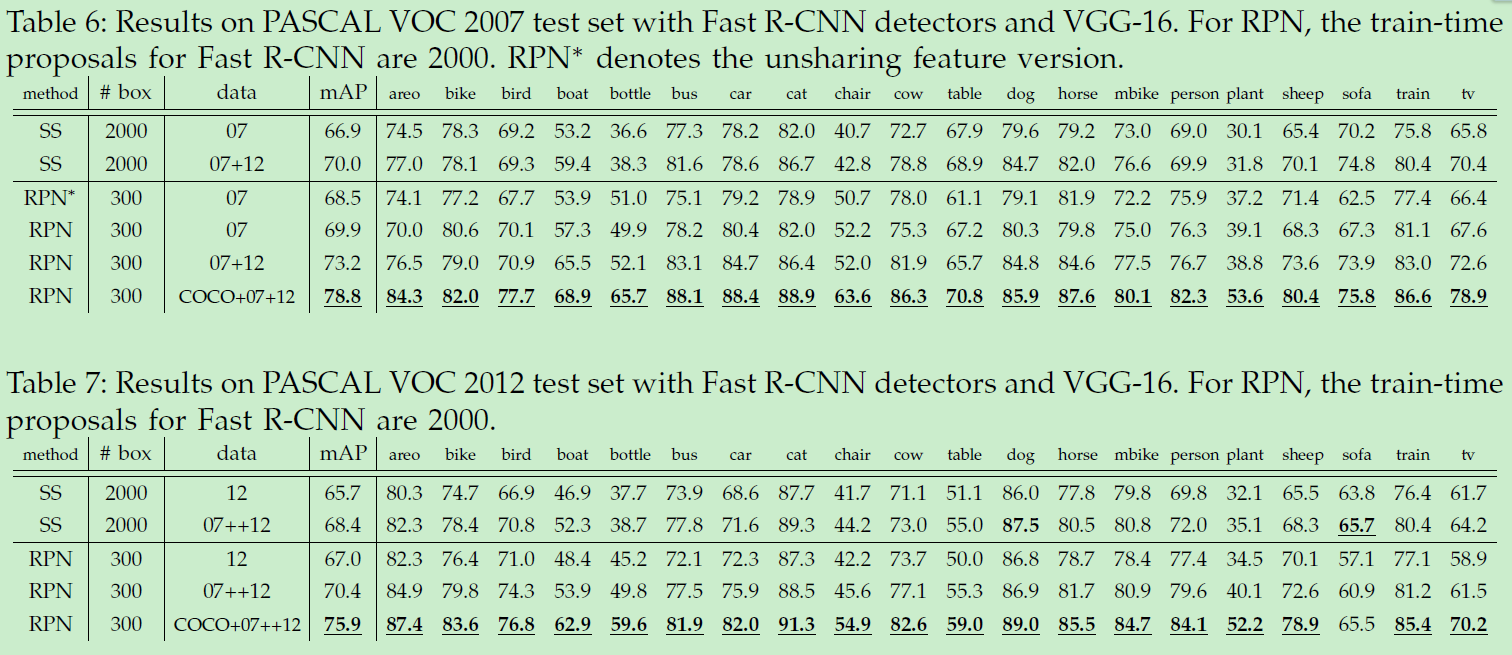


图19 stage1\_fast\_rcnn\_train.pt

之后的训练都是大同小异，不再赘述了。

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

**=====>Result**



**=====>Title**

**=====>Main Contributions / New Opinions**

**=====>Key Points**

**=====>Result**