知乎1

1、基于anchor和不基于anchor

除了yolov1和densebox之外的检测网络基本都是基于anchor的。

anchor相当于增加了模型的assumption，如果模型assumption跟数据集的分布一致，那么这个assumption会有利于框的回归，提高模型的准确性；但是如果这个assumption跟数据集分布不一致，它反而会伤害模型的准确性。所以基于anchor的模型，anchor的设置比较重要，anchor要与数据分布一致。

2、one stage与two stage

one stage主要有ssd和yolo系列，two stage有rcnn系列和基于ssd的refinedet。

虽然基于ssd的改进文章比yolo的多很多，给人一种ssd比yolo好的错觉，但是yolo各方面相比ssd并不差，ssd比yolo更受欢迎的原因我认为主要是ssd基于主流框架开发，便于改进；而yolo基于小众的darknet，darknet没多少人熟悉，大家也不愿意去熟悉这样的小众框架。

小众框架社区不成熟，遇到了问题不容易找到解决方案，而且学会了darknet只能玩yolo系列模型，学习成本太高，不值当，另外darknet只支持最简单的layer，稍微复杂的layer里面都没有，如果你想基于darknet搭一些当前最fancy的结构，可能会遇到困难。

one stage和two stage的区别。two stage相当于在one stage的后面再refine一下。ssd和v1之后的yolo跟faster rcnn系列框架里的rpn差不多。

one stage给人快但是准确性不太好的印象，two stage给人准确性高但是慢的印象。但是我认为假以时日，one stage和two stage会逐渐吸收彼此的长处，最后one stage可以做到和two stage一样准，two stage可以做到和one stage一样快。

如果以速度横轴，mAP为纵轴画点阵图，那么当前one stage模型主要分布在右下角，two stage模型主要分布在左上角；而以后的话，one stage模型在这个坐标系里每个点的附近一定会有一个two stage模型。

3、数据分布

模型其实就是一系列assumption的集合。这些assumption是我们对数据集分布的假设。

模型不是孤立的，模型跟数据分布是成对出现的，这也就是我们常说“没有最好的模型，只有最合适的模型”背后的道理。

深度学习模型的超强拟合能力让我们开始遗忘assumption的重要性。在传统机器学习时代，大家非常看重assumption，因为传统机器学习时代，模型拟合能力没那么强，模型的assumption一旦跟数据分布不一致，效果会很差。而深度学习时代，模型拟合能力太强，大多数时候我们不考虑assumption是否与数据分布契合，也能训练出一个不错的模型出来。

在深度学习刚火，到处一片蓝海，大家四处跑马圈地的时候，不关注assumption，粗放式地把模型往复杂了怼，的确是一种行之有效的方式。但是现在模型方面逐渐到了瓶颈，继续往复杂了怼，效果提升有限，这个时候我们可能需要从精细化入手，比如回到传统机器学习重视assumption的思路上。

以分类为例，现实场景里类别不平衡是常见现象，所以我们经常需要想一些办法对付类别不平衡，比如对数量少的类别升采样，或者对数量少类别的loss加更大的权重等等。

这个过程其实就是在处理数据分布跟assumption不一致的问题，普通分类模型的assumption是各个类别的样本数量差不多，但是实际数据集里各个类别的样本数量却差距较大，这里出现了gap。

以检测为例，提出ohem的原因也是从assumption角度入手。ohem是解决物体框和背景框之间的平衡问题。那么更进一步，检测里面各个类别之间平衡吗？

从assumption角度思考的话，会发现天空突然广阔很多，还有很多事情可做。我相信接下来两年会有越来越多的工作会从assumption入手，更甚，传统机器学习的思路也会慢慢地融入到深度学习领域。慢慢地，改改layer就能发文章的时代会逐渐远去。潮水褪去之后，既对传统机器学习了然于胸，又对深度学习有insight，还对具体领域（例如cv、nlp等等）有着扎实基础的人才能衣着光鲜地站在沙滩上，眺望远方，欣赏波涛汹涌的海浪。

目标检测总结（自己的一些见解）

目前object detection的工作可以粗略的分为两类： 1：使用region proposal的，目前是主流，比如RCNN、SPP-Net、Fast-RCNN、Faster-RCNN以及MSRA最近的工作R-FCN。 2：不使用region proposal的，YOLO，SSD。

从我这个渣渣的视野来看，这些工作都体现的一个趋势：如何让不同ROI之间尽量多的共享计算量，并充分利用CNN得到的特征，使得整个detection的速度变快。

具体说来，我们先回忆一下基于region proposal的方法的大致流程是什么样的：

1. 从待检测的图片中，提取出N个ROI，这里N远大于图片中真实object的个数。具体的方法有selective search、edge box以及最近流行起来的RPN。
2. 根据1中检测到的ROI，上CNN对图像进行feature extraction。
3. 对2中得到的feature进行分类，比如对于PSACAL VOC数据，就是一个21分类的问题（20个object class+background）。
4. boudningbox regression。

然后我们回到之前说的让ROI之间尽量多的共享计算量的问题。

RCNN对于每个ROI，都跑一遍CNN，即使这些ROI之间是有overlap的，显然有部分计算是重复的，所以SPP-net和fast rcnn就在这方面做了文章，具体做法是先用CNN抽取整张图的特征，然后利用ROI pooling抽取对应ROI的特征，使得不同ROI共享特征提取的计算量。结果就是原来我处理一张图像需要前向2000次CNN，现在只要前向一次就好了，极大的提升了计算速度。fast rcnn还通过multi-task loss实现了一个end to end 的系统，这里不是我们的重点。

fast-rcnn提出来之后，detection的性能瓶颈变成了计算region proposal。CPU实现的selective search处理一张图需要2秒钟，远大于GPU上CNN特征抽取的时间。Faster RCNN就是要解决这个问题，他的出发点是这样的：既然用CNN进行feature extraction这一步已经无法避免，那么我们为什么不更充分地利用得到的feature？具体来说，我们是不是可以直接用CNN得到的feature来进行region proposal，答案是肯定的。Faster RCNN将CNN得到的feature输入到一个两层网络（RPN），网络的输出就是region proposal。这样一来，region proposal的额外开销就只有一个两层网络。实验证明这样不仅速度变快，而且proposal的质量也更高了。

到目前为止，上面我们说的4个步骤中，第1步和第2步都可以通过前向一遍CNN来得到，所以前俩步都不再是速度的瓶颈。然后我们考虑第3步，假设我们用faster rcnn的RPN得到了300个region proposal，在预测的过程中，我们需要对300个region proposal去做分类，即我们要处理300个多分类问题，如果我们用one vs rest来实现多分类，每遍就是21个二分类线性svm（也可以用一个softmax-log loss的线性分类器，但是计算量和21个二分类svm差不多），这样的话每次预测就要跑6300个二分类器，即使是线性分类器，这个时间仍然是很大的，所以就有了R-FCN这个工作。具体来说，是先利用FCN进行类似semantic segmentation的计算，然后利用ROI对相应的区域进行average pooling，得到整个ROI关于21个类别的置信度。简单的说就是把分类这个过程也融合到网络的前向计算过程中，由于这个过程对于不同的ROI是共享的，所以比单独跑分类器要快好多。文章里还有一个position-sensitive的idea，也很有趣，不过感觉给我一种”这也能行“的感觉，应该是我少见多怪，理解不了大神的世界。

个人感觉object detection是一个比较考验insight以及”让一个idea真正能work的能力“的方向，不像semantic segmentation，后者现在的提升很多靠CRF，有陷入”图模型加圈“（传说中水论文三大法宝之一）的趋势，对数学要求比较高。

以上只是个人读paper的心得，可能作者们当时并不是这么想的。

作者：2014wzy 来源：CSDN 原文：https://blog.csdn.net/u014696921/article/details/53130852 版权声明：本文为博主原创文章，转载请附上博文链接！

关于物体检测，你想要的，这里都有（二）

Speed

从工业应用的角度来说，检测的速度也是至关重要的一环。从 R-CNN 到 Faster R-CNN ，检测速度越来越快，甚至经过一定优化，能够达到实时。 YOLO 和 SSD 这类 one-stage 检测器，速度可以达到几十帧，甚至上百帧。最近的一篇论文， Object detection at 200 Frames Per Second ，可以在精度较高的情况下，达到两百多帧的高速检测效果！如下图所示，可以看到，在 PASCAL VOC 2007 数据集上，文章提出的方法在速度上超过 YOLO 系列检测器，精度甚至能跟 Fast-RCNN 一战。

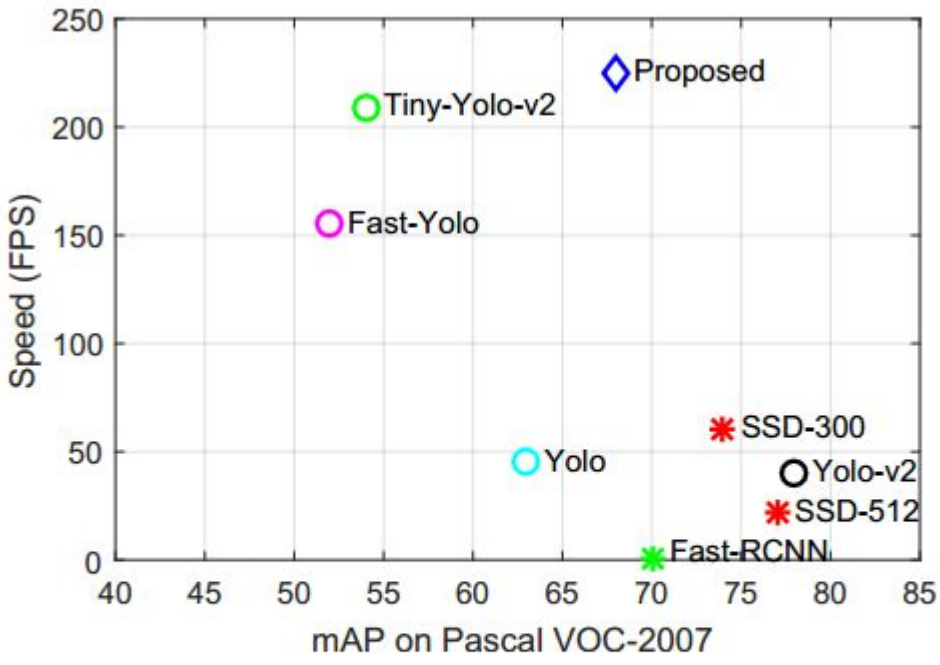


Figure 1: Speed and performance comparison of the proposed detector with other competing approaches. For SSD and Yolo-v2 we show results for more accurate models.

作者的目标很明确，要找到一个内存消耗少、速度快的高效物体检测器。为此，作者分析作为一个基于深度学习的检测框架的三要素：

(1) 网络架构 ；

(2) 损失函数；

(3) 训练数据。

因此，作者从以上三方面着手，不断优化改进，获得本文的高效检测器 D-YOLO 。

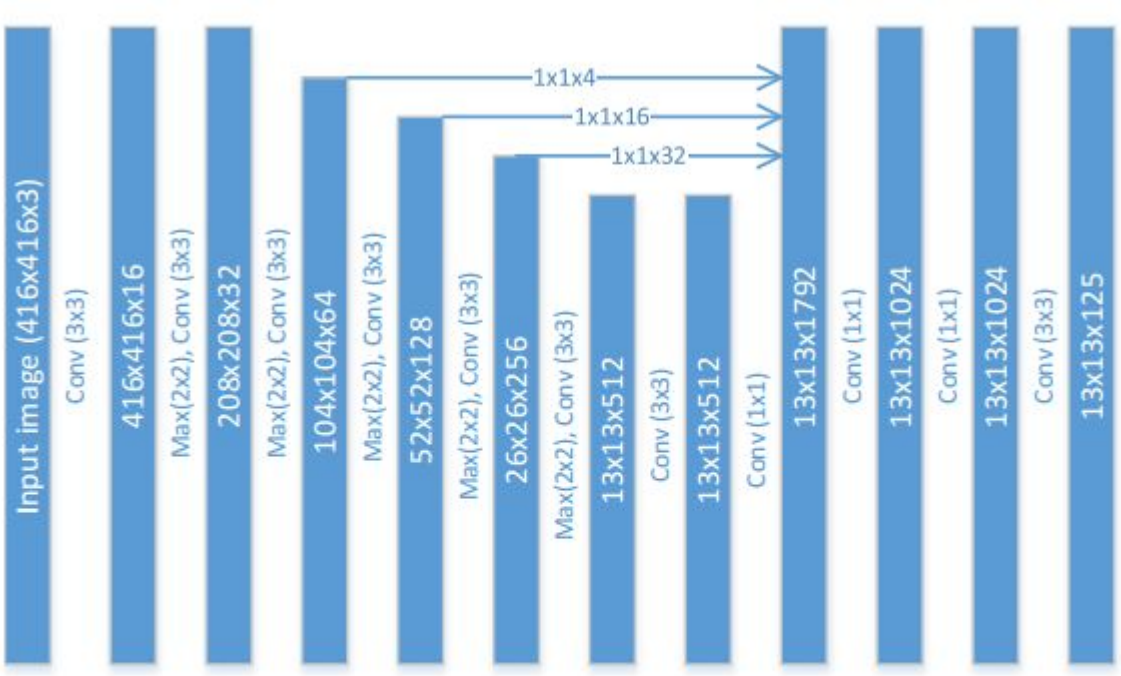


Figure 2: Base architecture of our detector. To keep architecture simple we limit the depth of the network, keep number of feature maps low and use a small filter kernel (3×3 or 1×1).

上图是作者提出的网络架构，作者主要根据特征融合和让网络变深变窄两大思想，在 Tiny-YOLO 网络基础上进行改进，得到 D-YOLO 的网络结构。

第一点是特征融合，这已经在 DenseNet 、 SSD 和 FPN 上验证了这种思路的有效性。作者为了将不同分辨率的特征图级联在一起，使用类似 YOLO 的 pass-through layer 的思路（作者发现 max pooling 会导致信息丢失，性能下降，弃用），将多个层的信息融合在一起，这里作者提到前面层的压缩比率可以大点。

如上图所示，作者将 104x104x64 、 52x52x128 和 26x26x256 各自分别通过 1x1x4 、 1x1x16 和 1x1x32 的卷积层，压缩得到 104x104x4 、 52x52x16 和 26x26x32 的输出特征图，每个特征图再 resize 成 13x13 分辨率大小的特征图，也就是 104x104x4 --> 13x13x256 ， 52x52x16 --> 13x13x256 , 26x26x32 --> 13x13x128 ，再和原来的特征图 concat 成1792维 13x13 大小特征图。这里可以计算下，原来的特征图应该是有 1792-256-256-128=1152 维。

除此之外， D-YOLO 还将后面几个特征图的维数减半（相较于 Tiny-YOLO ），网络变窄，增加了几个 1x1 的卷积层，让网络变得更深。

计算机视觉相关综述整理

概念：

MAP(mean average precision)：每一个类别都可以根据recall和precision绘制一条曲线，那么AP就是该曲线下的面积，而mAP是多个类别AP的平均值，这个值介于0到1之间，且越大越好。这个指标是目标检测算法最为重要的一个。

IOU：交并比——Intersection-over-Union，IoU。目标检测中使用的一个概念，是产生的候选框（candidate bound）与原标记框（ground truth bound）的交叠率，即它们的交集与并集的比值。最理想情况是完全重叠，即比值为1。

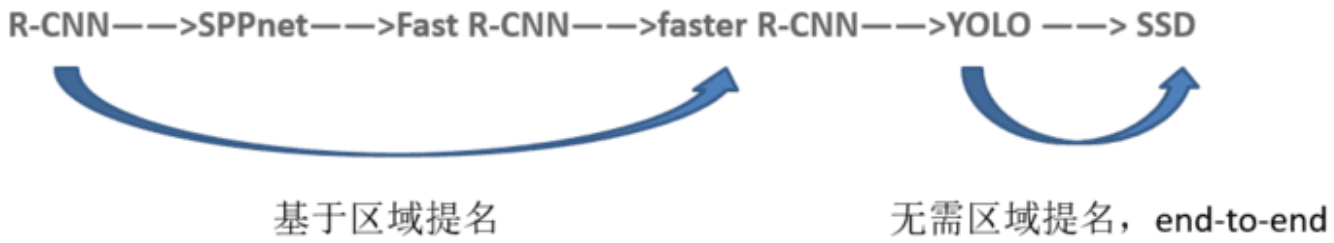
NMS(非极大值抑制)：目标检测算法一般会给出目标很多的粗略结果，对一个目标成百上千的粗略结果都进行调整肯定是不可行的。那么我们就需要对这些粗略结果先进行一个大体的挑选。挑选出其中最具代表性的结果。再对这些挑选后的结果进行调整，这样可以加快算法效率。消除多余的框，找到最佳的bbox。

边界框回归(Bounding-box regression)：将IOU较小的错误检测框进行调整，使得经过微调后的窗口跟Ground Truth 更接近 。

深度学习的方法：

two-stage detector：以R-CNN为代表；首先找到可能存在物体的候选框，然后对候选框进行类别预测和边界调整，检测准确度高但检测速度慢。 算法：R-CNN、Fast R-CNN、Faster R-CNN、R-FCN等。

one-stage detector：以YOLO为代表；是以回归的方式直接预测固定数量的候选框，检测准确度较低但检测速度快，能实时检测。 算法：YOLO、YOLOv2、SSD等。



OverFeat：第一个使用深度学习进行目标检测并取得很大进展的方法是纽约大学在 2013 年提出的 Overfeat ，他们提出了一个使用卷积神经网络(CNN)的多尺度滑动窗口算法。

R-CNN：在 Overfeat 提出不久，来自加州大学伯克利分校的 Ross Girshick 等人发表了 基于卷积神经网络特征的区域方法 (Regions with CNN features，R-CNN)，它在目标检测比赛上相比其他方法取得了 50%的性能提升。他们提出了一个三阶段的方法：

使用区域候选算法提取包含可能目标的区域(最流行的 选择性搜索算法) 使用 CNN 在每个区域上提取特征。 使用 支持向量机 对区域进行分类。 虽然该方法取得了很好的结果，但是训练阶段有很多问题。要训练网络，首先要生成训练数据集的候选区域，然后对每一个区域使用 CNN 提取特征(对于 Pascal 2012 训练数据集来说，通常需要生成大于 200GB 的中间文件)，最后训练 SVM 分类器。

Fast R-CNN：一年以后，Ross Girshick(微软亚洲研究院)发表了 Fast R-CNN ，这个方法迅速演化成一个纯深度学习的方法。与 R-CNN 相似，它也使用选择性搜索来生成候选区域，但与 R-CNN 不同的是，Fast R-CNN 在整张图上使用 CNN 来提取特征，然后在特征图上使用区域兴趣池化(Region of Interest，ROI)，并且最后用前馈神经网络来进行分类和回归。这个方法不仅快，而且由于RoI池化层和全连接层的存在，该模型可以进行端到端的求导，并且训练也更容易。最大的不足是该模型仍旧依赖选择性搜索(或者其他的区域候选算法)，这也成为了模型推理阶段的一个瓶颈。

YOLO：OLO 提出了一个简单的卷积神经网络方法，它取得了很好的结果，并且速度也非常快，第一次实现了实时的目标检测。先前提出的算法都是将检测问题转化为分类解决，而它将检测回归到回归方法，提高实时性能。

Faster R-CNN：Faster R-CNN 增加了一个所谓的“区域候选网络(Regio Proosal Network，RPN)”，试图摆脱搜索选择算法，从而让模型实现完全端到端的训练。简单地说，它的作用是根据“属于目标”的分数来输出可能目标。RoI 池化层和全连接层使用这些目标进行分类。

SSD 和 R-FCN：

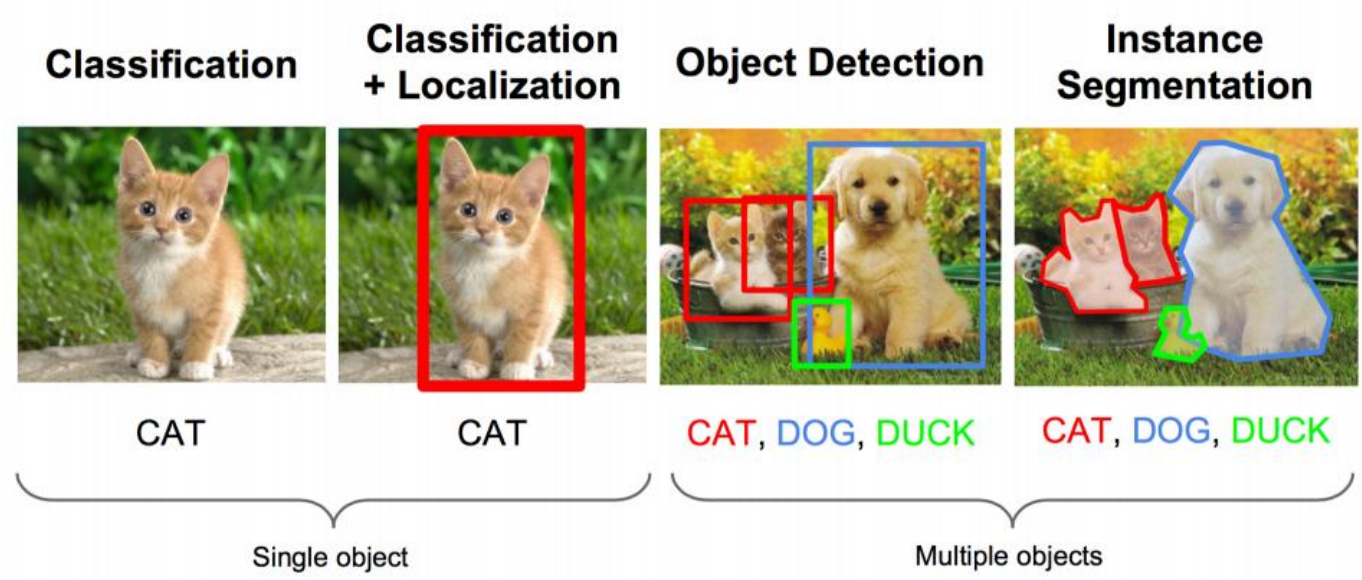
《单帧检测器》 (Single Shot Detector，SSD)，它在 YOLO 的基础上进行改良，通过使用多尺度的卷积特征图以达到更好的结果和速度。YOLO S×S的网格就是一个比较启发式的策略,难以检测小目标，借鉴了Faster R-CNN中的Anchor机制，使用了多尺度特征金字塔。

1. 基于深度学习的目标检测技术演进：R-CNN、Fast R-CNN、Faster R-CNN 2. 目标检测(Object Detection)的整理

object detection我的理解，就是在给定的图片中精确找到物体所在位置，并标注出物体的类别。object detection要解决的问题就是物体在哪里，是什么这整个流程的问题。然而，这个问题可不是那么容易解决的，物体的尺寸变化范围很大，摆放物体的角度，姿态不定，而且可以出现在图片的任何地方，更何况物体还可以是多个类别。

object detection技术的演进： RCNN->SppNET->Fast-RCNN->Faster-RCNN

从图像识别的任务说起 这里有一个图像任务： 既要把图中的物体识别出来，又要用方框框出它的位置。



上面的任务用专业的说法就是：图像识别+定位 图像识别（classification）： 输入：图片 输出：物体的类别 评估方法：准确率

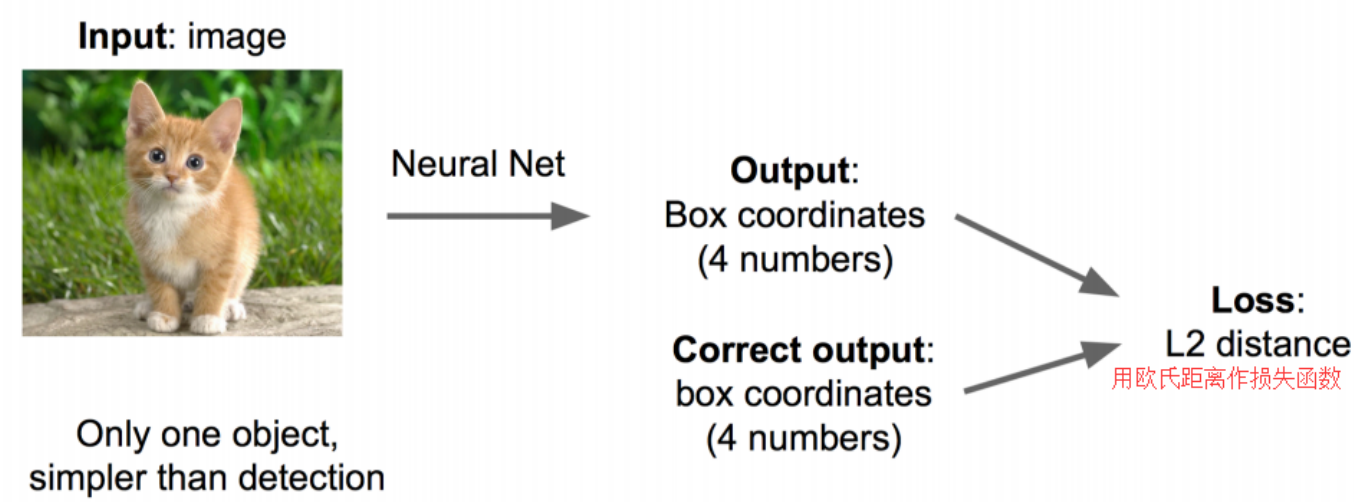
定位（localization）： 输入：图片 输出：方框在图片中的位置（x,y,w,h） 评估方法：检测评价函数 intersection-over-union (IOU)

卷积神经网络CNN已经帮我们完成了图像识别（判定是猫还是狗）的任务了，我们只需要添加一些额外的功能来完成定位任务即可。

定位的问题的解决思路有哪些?

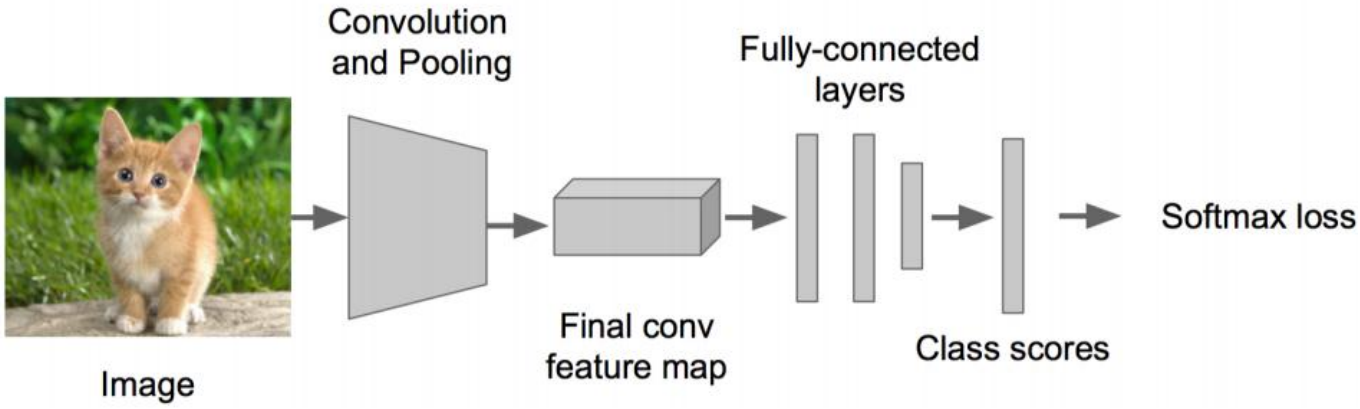
思路一：看做回归问题

看做回归问题，我们需要预测出（x,y,w,h）四个参数的值，从而得出方框的位置。



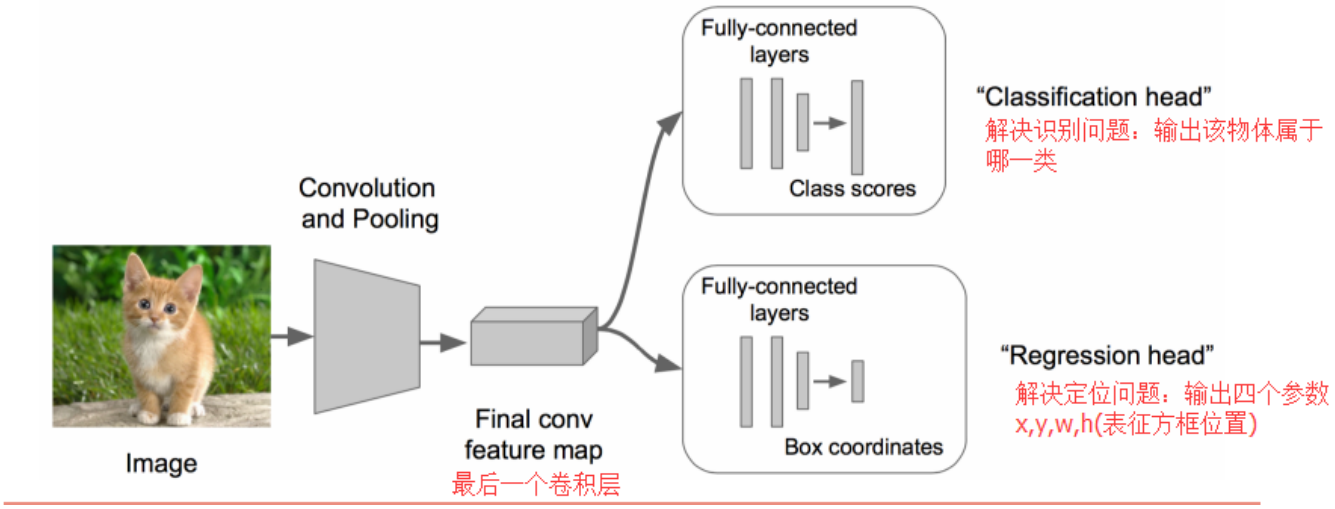
步骤**1:**

• 先解决简单问题， 搭一个识别图像的神经网络 • 在AlexNet VGG GoogleLenet上fine-tuning一下



步骤**2:**

• 在上述神经网络的尾部展开（也就说CNN前面保持不变，我们对CNN的结尾处作出改进：加了两个头：“分类头”和“回归头”） • 成为classification + regression模式



步骤3: • Regression那个部分用欧氏距离损失 • 使用SGD训练

步骤4: • 预测阶段把2个头部拼上 • 完成不同的功能

这里需要进行两次fine-tuning 第一次在ALexNet上做，第二次将头部改成regression head，前面不变，做一次fine-tuning

Regression的部分加在哪？

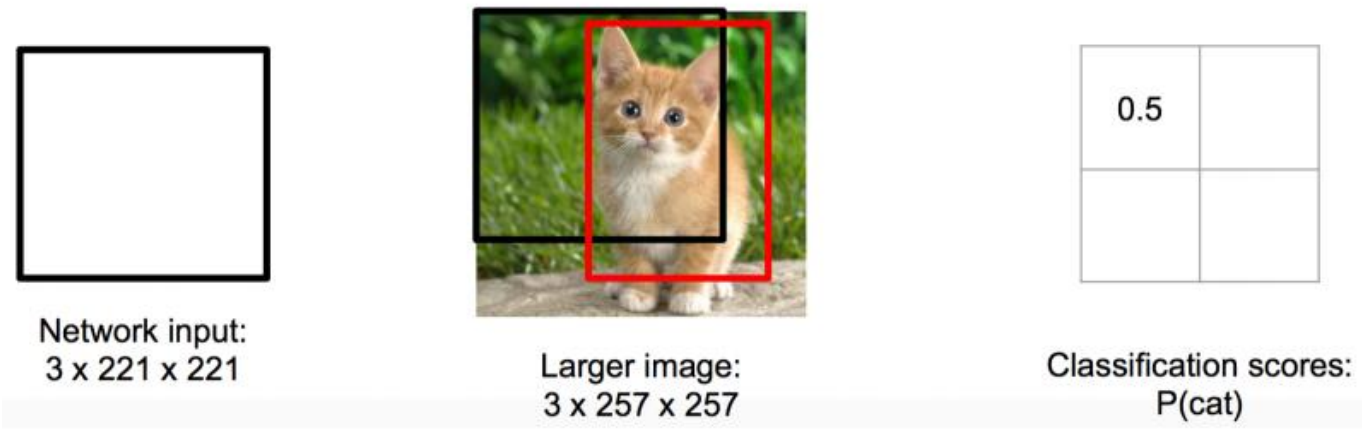
有两种处理方法： • 加在最后一个卷积层后面（如VGG） • 加在最后一个全连接层后面（如R-CNN）

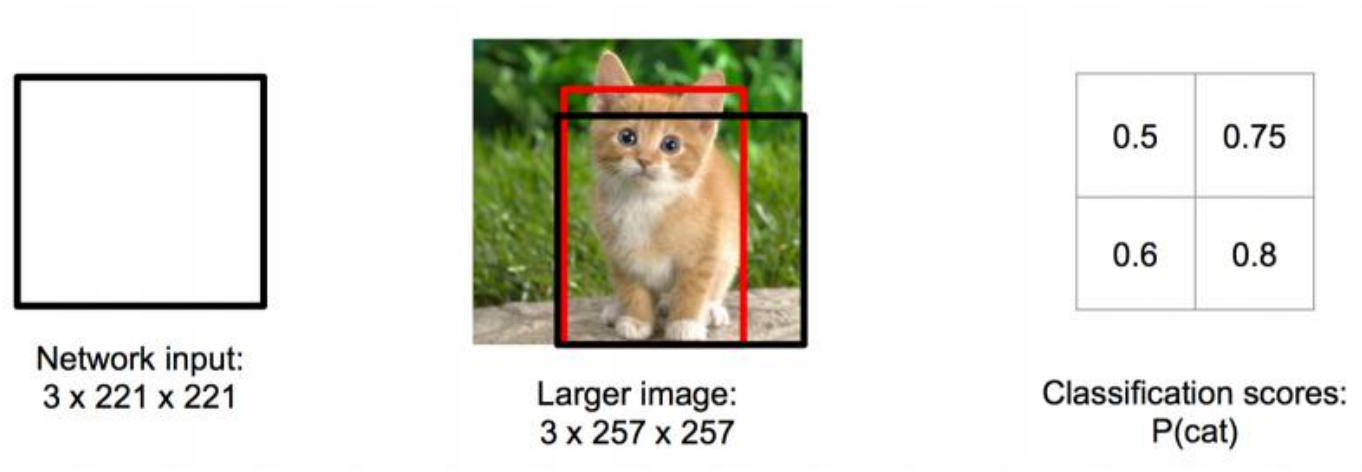
regression太难做了，应想方设法转换为classification问题。 regression的训练参数收敛的时间要长得多，所以上面的网络采取了用classification的网络来计算出网络共同部分的连接权值。

思路二：取图像窗口

• 还是刚才的classification + regression思路 • 咱们取不同的大小的“框” • 让框出现在不同的位置，得出这个框的判定得分 • 取得分最高的那个框

左上角的黑框：得分0.5 右上角的黑框：得分0.75 左下角的黑框：得分0.6 右下角的黑框：得分0.8





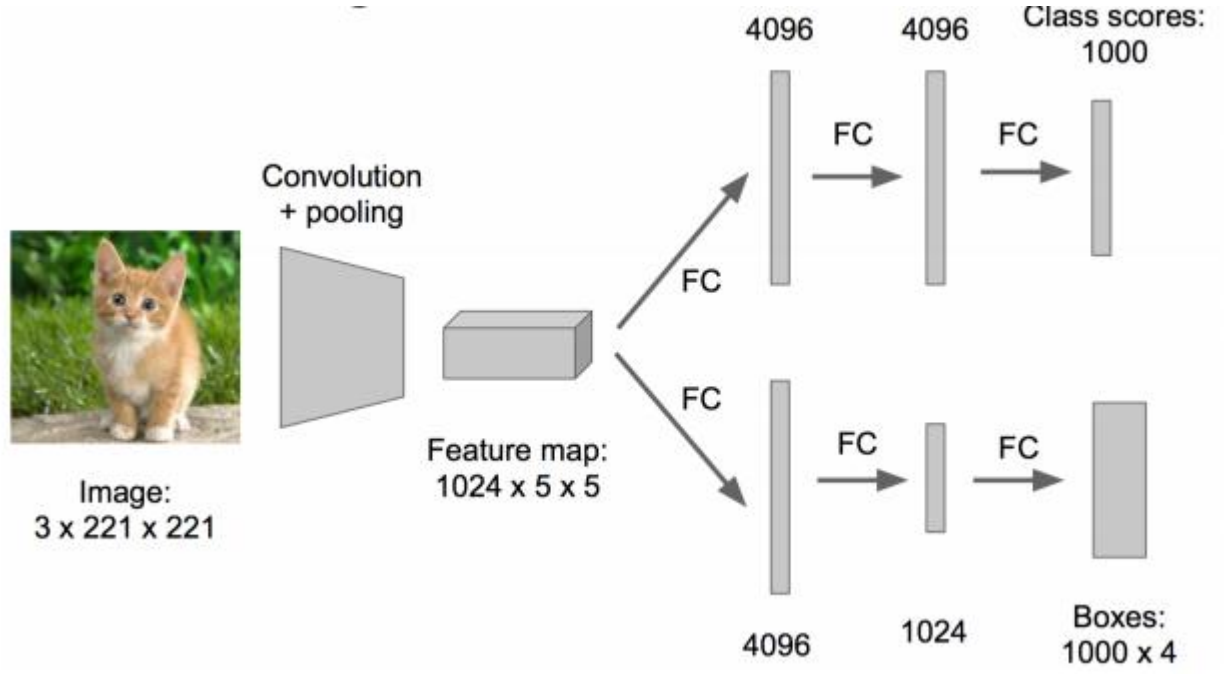
根据得分的高低，我们选择了右下角的黑框作为目标位置的预测。 注：有的时候也会选择得分最高的两个框，然后取两框的交集作为最终的位置预测。

疑惑：框要取多大？ 取不同的框，依次从左上角扫到右下角。非常粗暴啊。

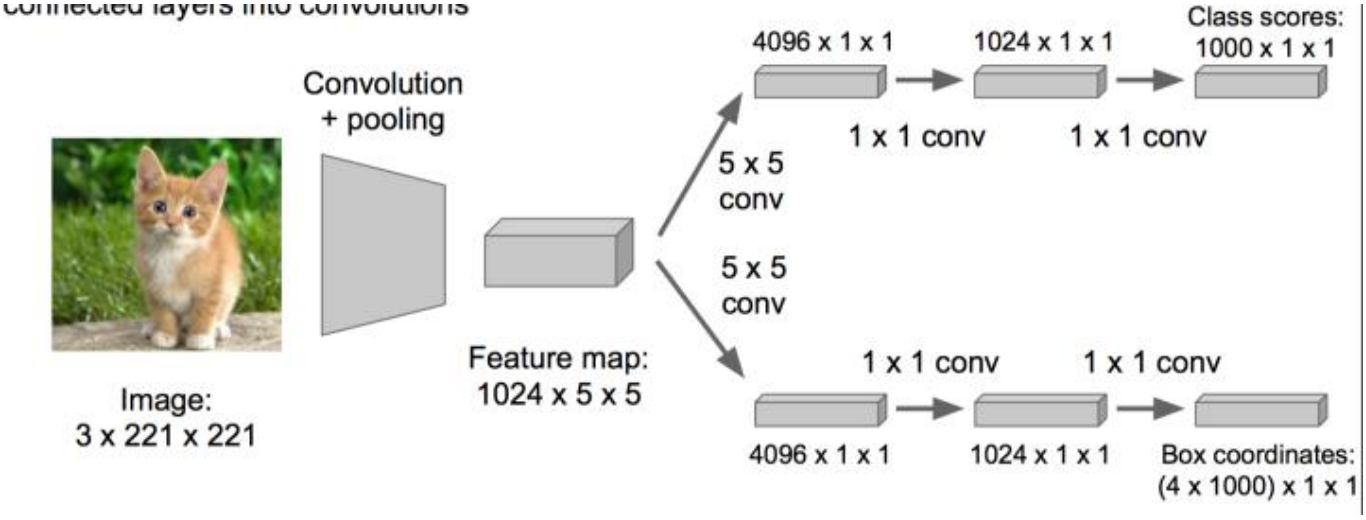
总结一下思路：

对一张图片，用各种大小的框（遍历整张图片）将图片截取出来，输入到CNN，然后CNN会输出这个框的得分（classification）以及这个框图片对应的x,y,h,w（regression）。

这方法实在太耗时间了，做个优化。 原来网络是这样的：



优化成这样：把全连接层改为卷积层，这样可以提提速。



物体检测（Object Detection） 当图像有很多物体怎么办的？难度可是一下暴增啊。

那任务就变成了：多物体识别+定位多个物体 那把这个任务看做分类问题？ 看成分类问题有何不妥？ • 你需要找很多位置， 给很多个不同大小的框 • 你还需要对框内的图像分类 • 当然， 如果你的GPU很强大， 恩， 那加油做吧…

看做classification， 有没有办法优化下？我可不想试那么多框那么多位置啊！ 有人想到一个好方法： 找出可能含有物体的框（也就是候选框，比如选1000个候选框），这些框之间是可以互相重叠互相包含的，这样我们就可以避免暴力枚举的所有框了。

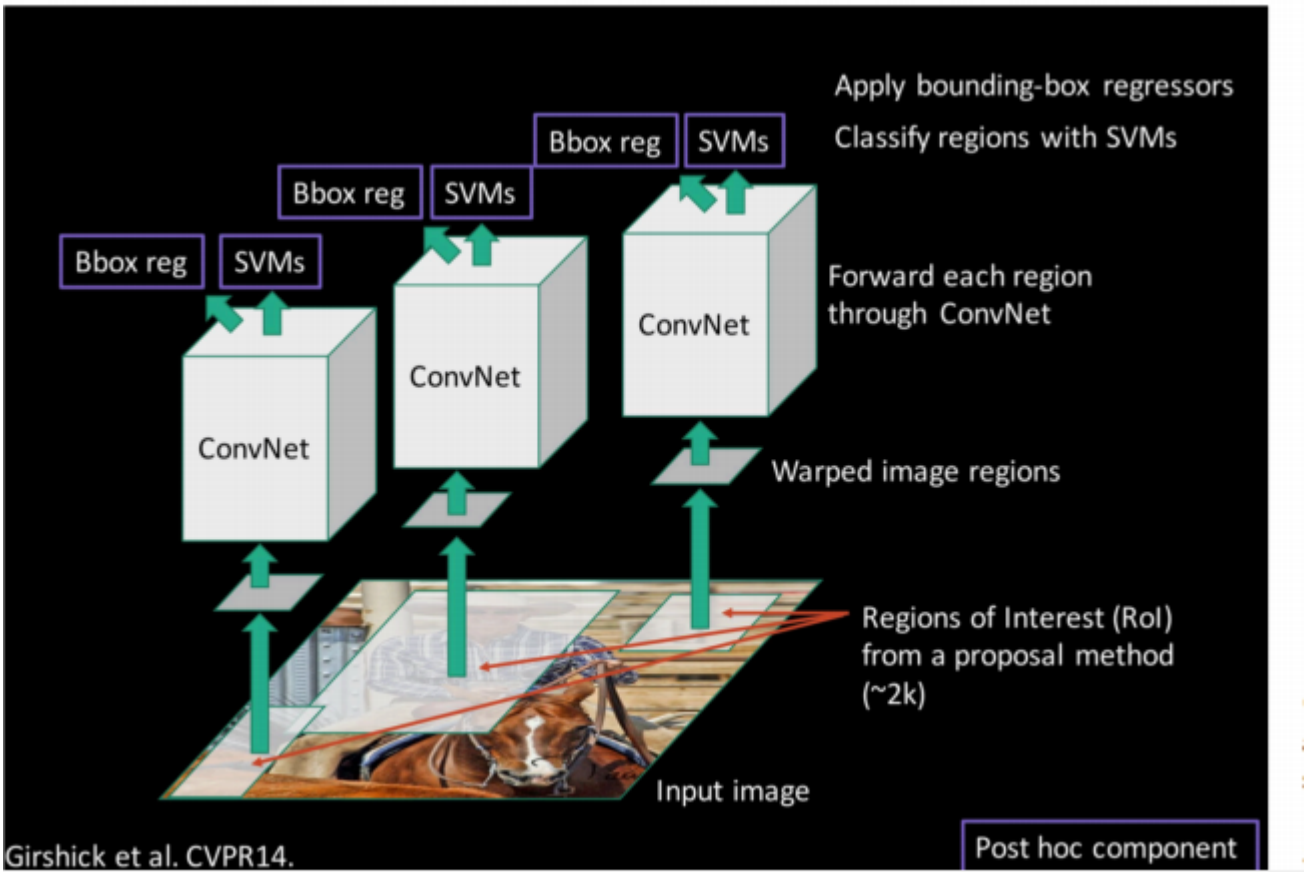
大牛们发明好多选定候选框的方法，比如EdgeBoxes和Selective Search。 以下是各种选定候选框的方法的性能对比。

Method	Approach	Outputs Segments	Outputs Score	Control #proposals	Time (sec.)	Repea-tability	Recall Results	Detection Results
Bing [18]	Window scoring		✓	✓	0.2	⋆⋆⋆	⋆	·
CPMC [19]	Grouping	✓	✓	✓	250	-	⋆⋆	⋆
EdgeBoxes [20]	Window scoring		✓	✓	0.3	⋆⋆	⋆⋆⋆	⋆⋆⋆
Endres [21]	Grouping	✓	✓	✓	100	-	⋆⋆⋆	⋆⋆
Geodesic [22]	Grouping	✓		✓	1	⋆	⋆⋆⋆	⋆⋆
MCG [23]	Grouping	✓	✓	✓	30	⋆	⋆⋆⋆	⋆⋆⋆
Objectness [24]	Window scoring		✓	✓	3	·	⋆	·
Rahtu [25]	Window scoring		✓	✓	3	·	·	⋆
RandomizedPrim's [26]	Grouping	✓		✓	1	⋆	⋆	⋆⋆
Rantalankila [27]	Grouping	✓		✓	10	⋆⋆	·	⋆⋆
Rigor [28]	Grouping	✓		✓	10	⋆	⋆⋆	⋆⋆
SelectiveSearch [29]	Grouping	✓	✓	✓	10	⋆⋆	⋆⋆⋆	⋆⋆⋆
Gaussian				✓	0	·	·	⋆
SlidingWindow				✓	0	⋆⋆⋆	·	·
Superpixels		✓			1	⋆	·	·
Uniform				✓	0	·	·	·

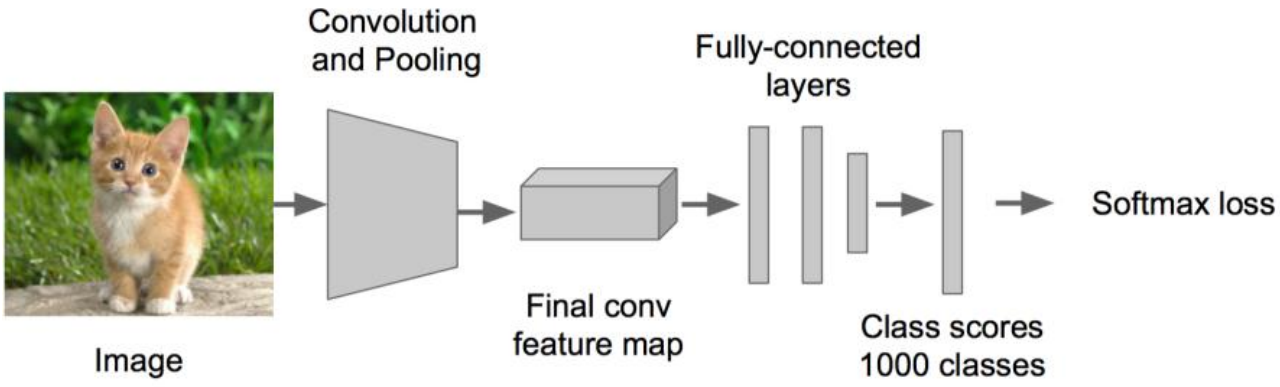
有一个很大的疑惑，提取候选框用到的算法“选择性搜索”到底怎么选出这些候选框的呢？那个就得好好看看它的论文了，这里就不介绍了。

R-CNN横空出世

基于以上的思路，RCNN的出现了。

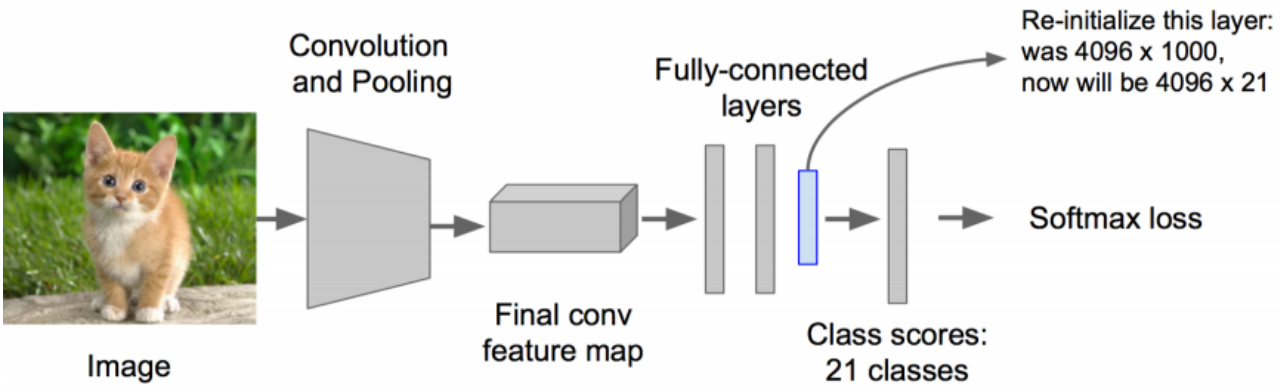


步骤一：训练（或者下载）一个分类模型（比如**AlexNet**）



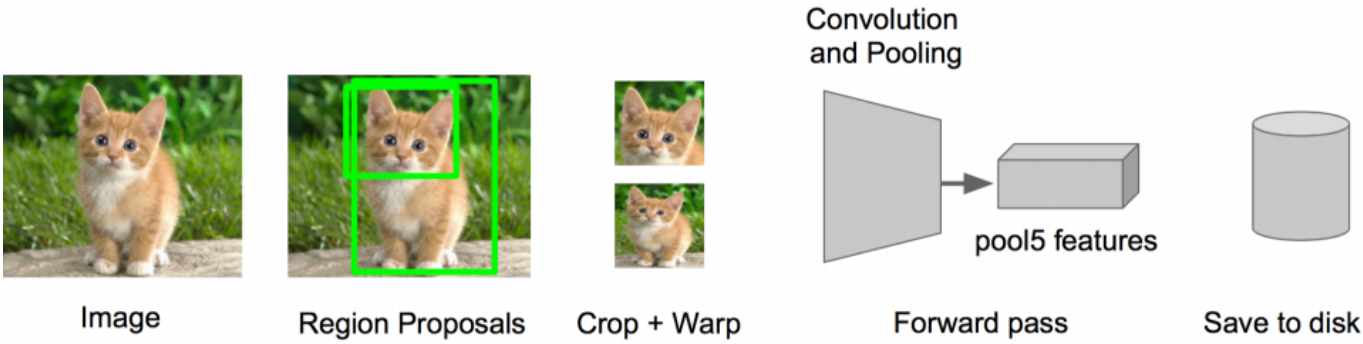
步骤二：对该模型做**fine-tuning**

• 将分类数从1000改为20　　• 去掉最后一个全连接层



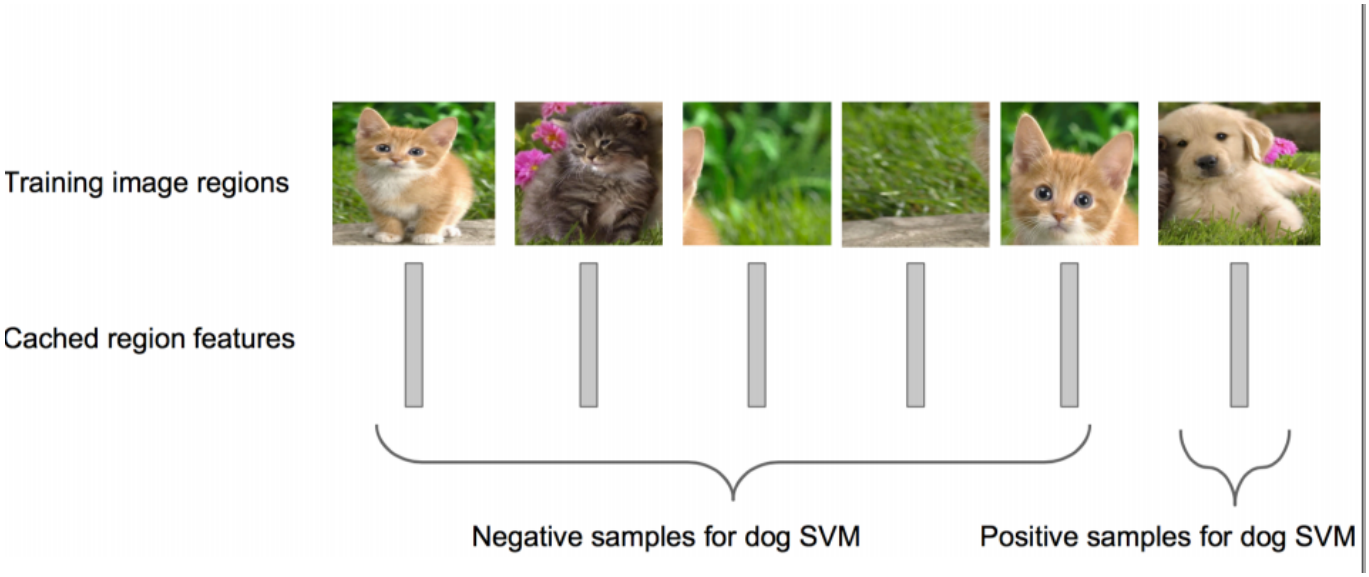
步骤三：特征提取

• 提取图像的所有候选框（选择性搜索）　　• 对于每一个区域：修正区域大小以适合CNN的输入，做一次前向运算，将第五个池化层的输出（就是对候选框提取到的特征）存到硬盘



步骤四：训练一个**SVM**分类器（二分类）来判断这个候选框里物体的类别

每个类别对应一个SVM，判断是不是属于这个类别，是就是positive，反之nagative 比如下图，就是狗分类的SVM



步骤五：使用回归器精细修正候选框位置：对于每一个类，训练一个线性回归模型去判定这个框是否框得完美。

RCNN的进化中SPP Net的思想对其贡献很大，这里也简单介绍一下SPP Net。

R-CNN缺点：

> (1) 训练分为多个阶段，步骤繁琐： 微调网络+训练SVM+训练边框回归器
> (2) 训练耗时，占用磁盘空间大：5000张图像产生几百G的特征文件
> (3) 速度慢： 使用GPU, VGG16模型处理一张图像需要47s。
>
> (4) 测试速度慢：每个候选区域需要运行整个前向CNN计算
> (5) SVM和回归是事后操作：在SVM和回归过程中CNN特征没有被学习更新
> 针对速度慢的这个问题，SPP-NET给出了很好的解决方案。

SPP Net

先看一下R-CNN为什么检测速度这么慢，一张图都需要47s！仔细看下R-CNN框架发现，对图像提完Region Proposal（2000个左右）之后将每个Proposal当成一张图像进行后续处理(CNN提特征+SVM分类)，实际上对一张图像进行了2000次提特征和分类的过程！这2000个Region Proposal不都是图像的一部分吗，那么我们完

全可以对图像提一次卷积层特征，然后只需要将Region Proposal在原图的位置映射到卷积层特征图上，这样对于一张图像我们只需要提一次卷积层特征，然后将每个Region Proposal的卷积层特征输入到全连接层做后续操作。（对于CNN来说，大部分运算都耗在卷积操作上，这样做可以节省大量时间）。

现在的问题是每个Region Proposal的尺度不一样，直接这样输入全连接层肯定是不行的，因为全连接层输入必须是固定的长度。SPP-NET恰好可以解决这个问题。

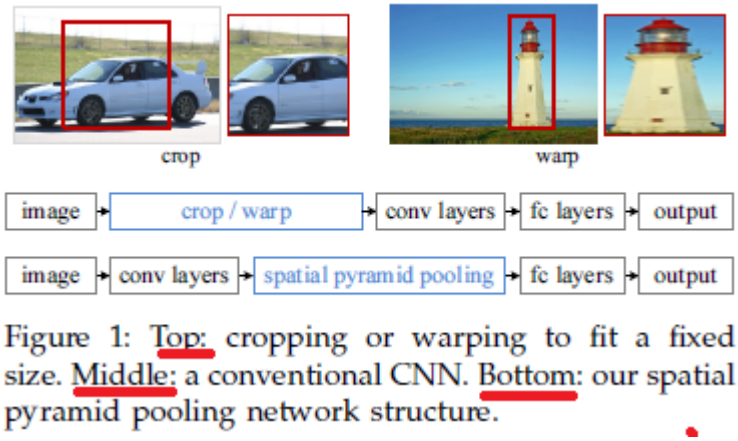


Figure 1: Top: cropping or warping to fit a fixed size. Middle: a conventional CNN. Bottom: our spatial pyramid pooling network structure.

由于传统的CNN限制了输入必须固定大小（比如AlexNet是224x224），所以在实际使用中往往需要对原图片进行crop或者warp的操作： - crop：截取原图片的一个固定大小的patch - warp：将原图片的ROI缩放到一个固定大小的patch

无论是crop还是warp，都无法保证在不失真的情况下将图片传入到CNN当中：

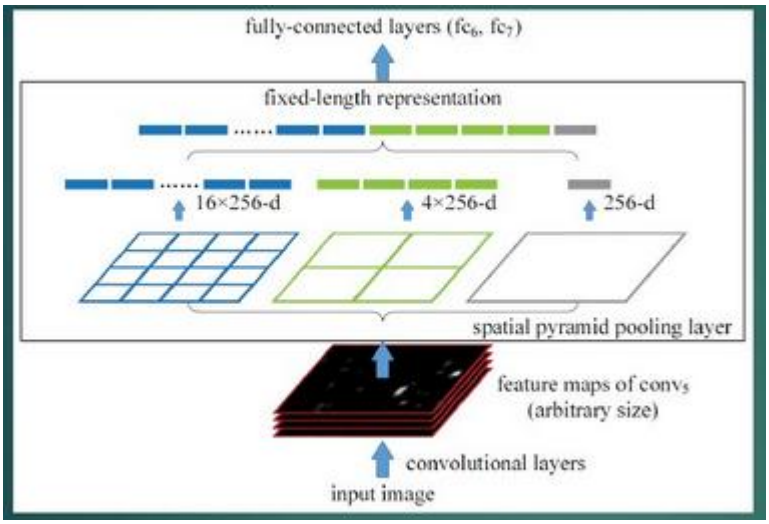
- crop：物体可能会产生截断，尤其是长宽比大的图片。
- warp：物体被拉伸，失去“原形”，尤其是长宽比大的图片

SPP为的就是解决上述的问题，做到的效果为：不管输入的图片是什么尺度，都能够正确的传入网络。 具体思路为：CNN的卷积层是可以处理任意尺度的输入的，只是在全连接层处有限制尺度——换句话说，如果找到一个方法，在全连接层之前将其输入限制到等长，那么就解决了这个问题。

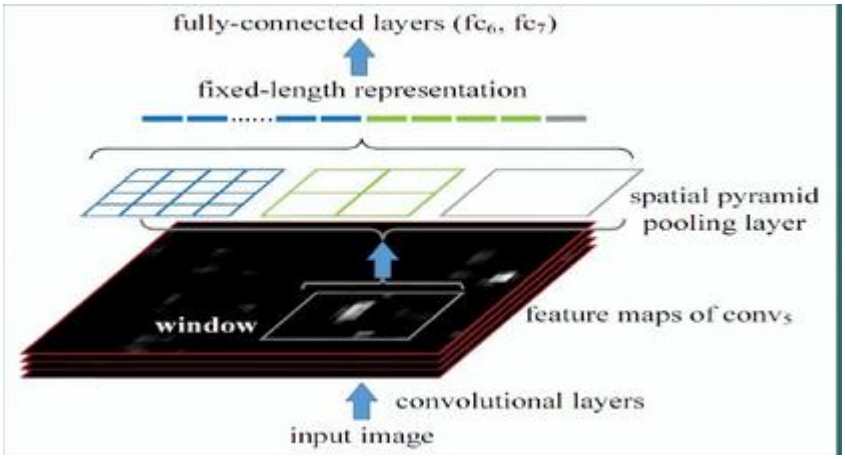
SPP：Spatial Pyramid Pooling（空间金字塔池化） 它的特点有两个:

1.结合空间金字塔方法实现CNNs的对尺度输入。 一般CNN后接全连接层或者分类器，他们都需要固定的输入尺寸，因此不得不对输入数据进行crop或者warp，这些预处理会造成数据的丢失或几何的失真。SPP Net的第一个贡献就是将金字塔思想加入到CNN，实现了数据的多尺度输入。

如下图所示，在卷积层和全连接层之间加入了SPP layer。此时网络的输入可以是任意尺度的，在SPP layer中每一个pooling的filter会根据输入调整大小，而SPP的输出尺度始终是固定的。



2.只对原图提取一次卷积特征 在R-CNN中，每个候选框先resize到统一大小，然后分别作为CNN的输入，这样是很低效的。 所以SPP Net根据这个缺点做了优化：只对原图进行一次卷积得到整张图的feature map，然后找到每个候选框zaifeature map上的映射patch，将此patch作为每个候选框的卷积特征输入到SPP layer和之后的层。节省了大量的计算时间，比R-CNN有一百倍左右的提速。



如果原图输入是224x224，对于conv5出来后的输出，是13x13x256的，可以理解成有256个这样的filter，每个filter对应一张13x13的activation map。如果像上图那样将activation map pooling成4x4 2x2 1x1三张子图，做max pooling后，出来的特征就是固定长度的(16+4+1)x256那么多的维度了。如果原图的输入不是224x224，出来的特征依然是(16+4+1)x256；直觉地说，可以理解成将原来固定大小为(3x3)窗口的pool5改成了自适应窗口大小，窗口的大小和activation map成比例，保证了经过pooling后出来的feature的长度是一致的。

使用SPP-NET相比于R-CNN可以大大加快目标检测的速度，但是依然存在着很多问题： (1) 训练分为多个阶段，步骤繁琐: 微调网络+训练SVM+训练训练边框回归器 (2) SPP-NET在微调网络的时候固定了卷积层，只对全连接层进行微调，而对于一个新的任务，有必要对卷积层也进行微调。（分类的模型提取的特征更注重高层语义，而目标检测任务除了语义信息还需要目标的位置信息） 针对这两个问题，RBG又提出Fast R-CNN, 一个精简而快速的目标检测框架。

Fast R-CNN

SPP Net真是个好方法，R-CNN的进阶版Fast R-CNN就是在RCNN的基础上采纳了SPP Net方法，对RCNN作了改进，使得性能进一步提高。

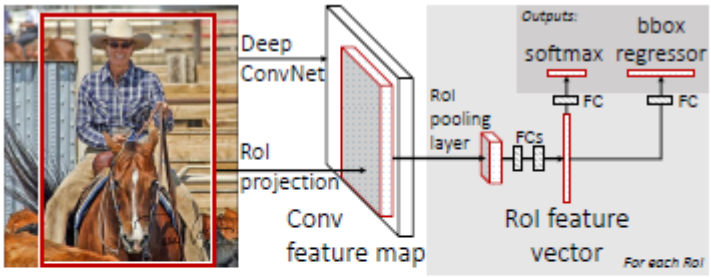


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

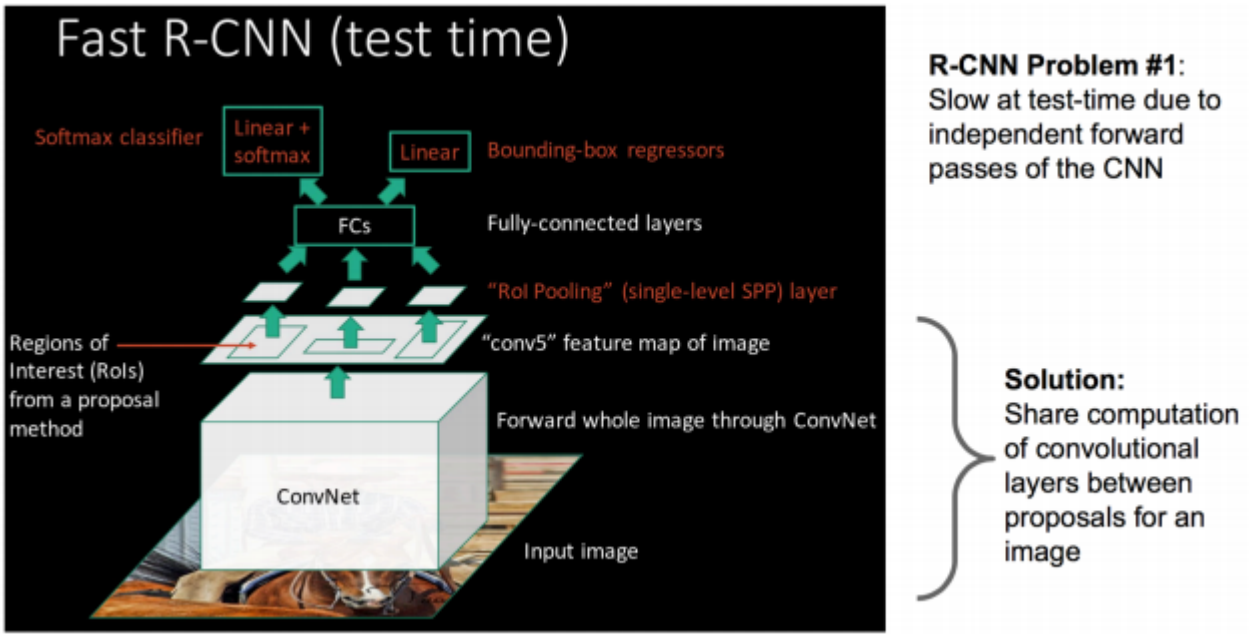
R-CNN与Fast RCNN的区别有哪些呢？ 先说RCNN的缺点：即使使用了selective search等预处理步骤来提取潜在的bounding box作为输入，但是RCNN仍会有严重的速度瓶颈，原因也很明显，就是计算机对所有region进行特征提取时会有重复计算，Fast-RCNN正是为了解决这个问题诞生的。

与R-CNN框架图对比，可以发现主要有两处不同：一是最后一个卷积层后加了一个ROI pooling layer，二是损失函数使用了多任务损失函数(multi-task loss)，将边框回归直接加入到CNN网络中训练。

(1) ROI pooling layer实际上是SPP-NET的一个精简版，SPP-NET对每个proposal使用了不同大小的金字塔映射，而ROI pooling layer只需要下采样到一个7x7的特征图。对于VGG16网络conv5_3有512个特征图，这样所有region proposal对应了一个77512维度的特征向量作为全连接层的输入。

(2) R-CNN训练过程分为了三个阶段，而Fast R-CNN直接使用softmax替代SVM分类，同时利用多任务损失函数边框回归也加入到了网络中，这样整个的训练过程是端到端的(除去region proposal提取阶段)

(3) Fast R-CNN在网络微调的过程中，将部分卷积层也进行了微调，取得了更好的检测效果。



大牛提出了一个可以看做单层sppnet的网络层，叫做ROI Pooling，这个网络层可以把不同大小的输入映射到一个固定尺度的特征向量，而我们知道，conv、pooling、relu等操作都不需要固定size的输入，因此，在原始图

片上执行这些操作后，虽然输入图片size不同导致得到的feature map尺寸也不同，不能直接接到一个全连接层进行分类，但是可以加入这个神奇的ROI Pooling层，对每个region都提取一个固定维度的特征表示，再通过正常的softmax进行类型识别。另外，之前RCNN的处理流程是先提proposal，然后CNN提取特征，之后用SVM分类器，最后再做bbox regression，而在Fast-RCNN中，作者巧妙的把bbox regression放进了神经网络内部，与region分类和并成为了一个multi-task模型，实际实验也证明，这两个任务能够共享卷积特征，并相互促进。
Fast-RCNN很重要的一个贡献是成功的让人们看到了Region Proposal+CNN这一框架实时检测的希望，原来多类检测真的可以在保证准确率的同时提升处理速度，也为后来的Faster-RCNN做下了铺垫。

画一画重点： R-CNN有一些相当大的缺点（把这些缺点都改掉了，就成了Fast R-CNN）。 大缺点：由于每一个候选框都要独自经过CNN，这使得花费的时间非常多。 解决：共享卷积层，现在不是每一个候选框都当做输入进入CNN了，而是输入一张完整的图片，在第五个卷积层再得到每个候选框的特征

原来的方法：许多候选框（比如两千个）-->CNN-->得到每个候选框的特征-->分类+回归 现在的方法：一张完整图片-->CNN-->得到每张候选框的特征-->分类+回归

所以容易看见，Fast RCNN相对于RCNN的提速原因就在于：不过不像RCNN把每个候选区域给深度网络提特征，而是整张图提一次特征，再把候选框映射到conv5上，而SPP只需要计算一次特征，剩下的只需要在conv5层上操作就可以了。 在性能上提升也是相当明显的：

		R-CNN	Fast R-CNN
Faster!	Training Time:	84 hours	**9.5 hours**
	(Speedup)	1x	**8.8x**
FASTER!	Test time per image	47 seconds	**0.32 seconds**
	(Speedup)	1x	**146x**

1）Fast R-CNN优点：

Fast R-CNN融合了R-CNN和SPP-NET的精髓，并且引入多任务损失函数，使整个网络的训练和测试变得十分方便。在Pascal VOC2007训练集上训练，在VOC2007测试的结果为66.9%(mAP)，如果使用VOC2007+2012训练集训练，在VOC2007上测试结果为70%（数据集的扩充能大幅提高目标检测性能）。使用VGG16每张图像总共需要3s左右。

2）Fast R-CNN 缺点：

Region Proposal的提取使用selective search，目标检测时间大多消耗在这上面（提Region Proposal 2~3s，而提特征分类只需0.32s），无法满足实时应用，而且并没有实现真正意义上的端到端训练测试（region proposal使用selective search先提取处来）。那么有没有可能直接使用CNN直接产生Region Proposal并对其分类？
Faster R-CNN框架就是符合这样需要的目标检测框架。

Faster R-CNN

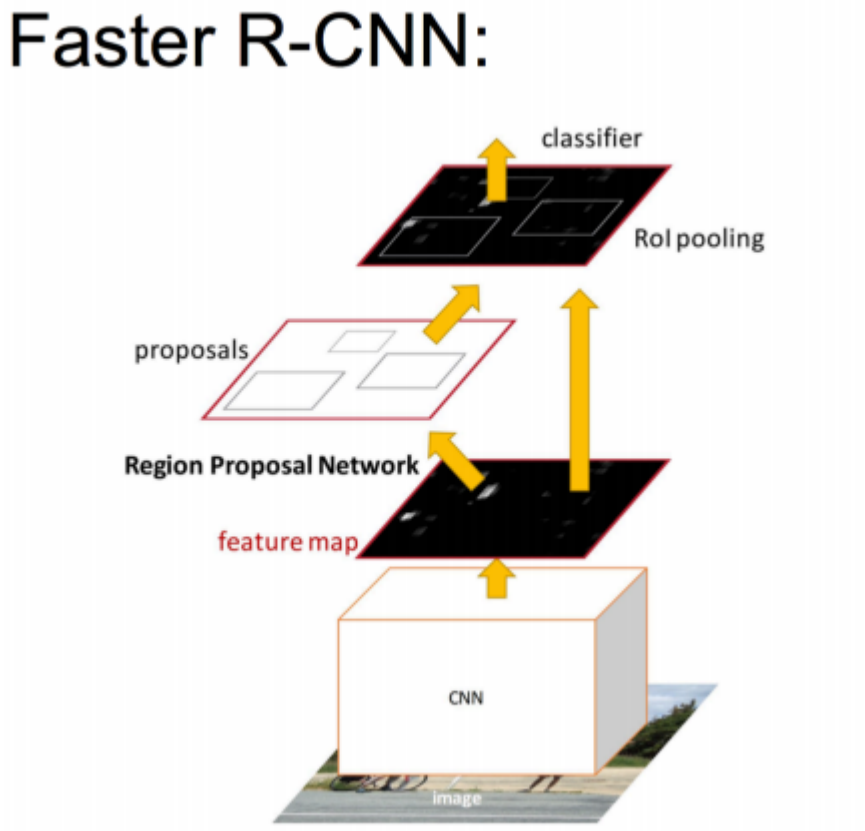
Fast R-CNN存在的问题：存在瓶颈：选择性搜索，找出所有的候选框，这个也非常耗时。那我们能不能找出一个更加高效的方法来求出这些候选框呢？ 解决：加入一个提取边缘的神经网络，也就说找到候选框的工作也交给神经网络来做了。 做这样的任务的神经网络叫做Region Proposal Network(RPN)。

在Region Proposal + CNN分类的这种目标检测框架中，Region Proposal质量好坏直接影响到目标检测任务的精度。如果找到一种方法只提取几百个或者更少的高质量的假选窗口，而且召回率很高，这不但能加快目标检测速度，还能提高目标检测的性能（假阳例少）。RPN(Region Proposal Networks)网络应运而生。

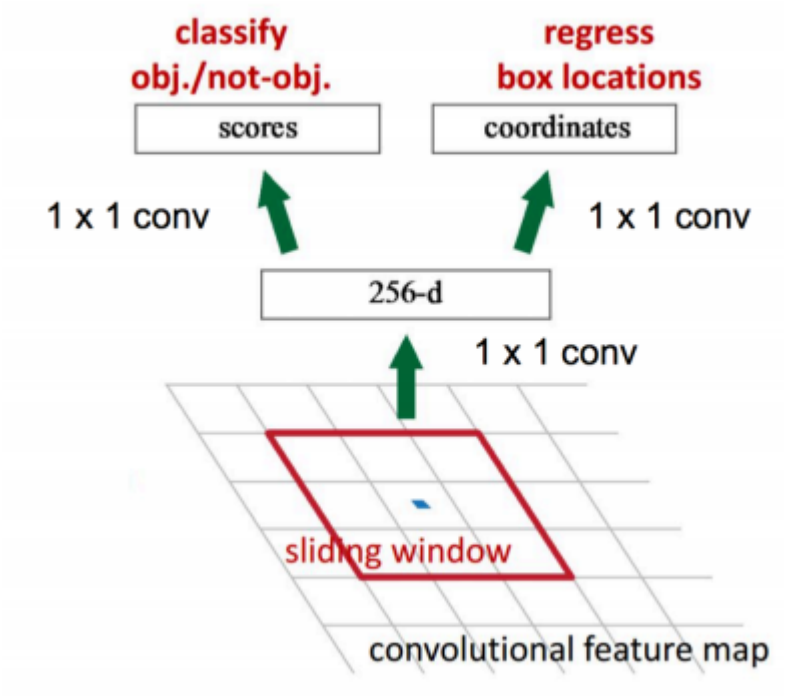
1）RPN的核心思想

> 是使用卷积神经网络直接产生Region Proposal，使用的方法本质上就是滑动窗口。RPN的设计比较巧妙，RPN只需在最后的卷积层上滑动一遍，因为Anchor机制和边框回归可以得到多尺度多长宽比的Region Proposal。

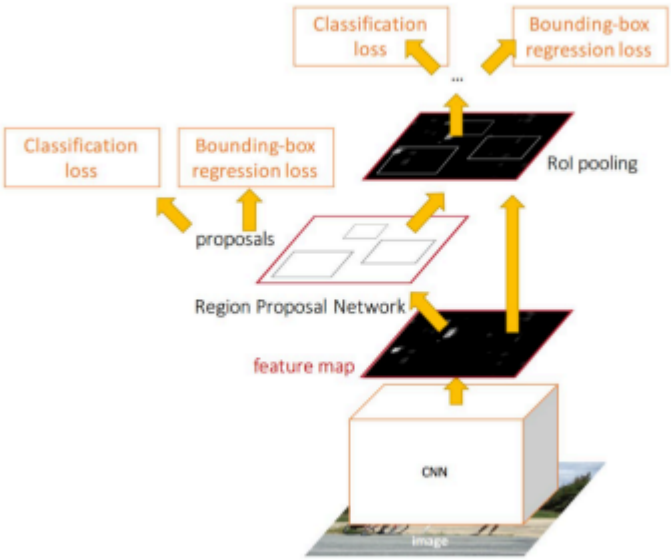
具体做法： • 将RPN放在最后一个卷积层的后面 • RPN直接训练得到候选区域



RPN简介： • 在feature map上滑动窗口 • 建一个神经网络用于物体分类+框位置的回归 • 滑动窗口的位置提供了物体的大体位置信息 • 框的回归提供了框更精确的位置



一种网络，四个损失函数; • RPN calssification(anchor good.bad) • RPN regression(anchor->propoasal) • Fast R-CNN classification(over classes) • Fast R-CNN regression(proposal ->box)



速度对比

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	**0.2 seconds**
(Speedup)	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**66.9**

Faster R-CNN的主要贡献是设计了提取候选区域的网络RPN，代替了费时的选择性搜索，使得检测速度大幅提高。

最后总结一下各大算法的步骤： RCNN 1. 在图像中确定约1000-2000个候选框 (使用选择性搜索) 2. 每个候选框内图像块缩放至相同大小，并输入到CNN内进行特征提取 3. 对候选框中提取出的特征，使用分类器判别是否属于一个特定类 4. 对于属于某一特征的候选框，用回归器进一步调整其位置

Fast RCNN 1. 在图像中确定约1000-2000个候选框 (使用选择性搜索) 2. 对整张图片输进CNN，得到feature map 3. 找到每个候选框在feature map上的映射patch，将此patch作为每个候选框的卷积特征输入到SPP layer和之后的层 4. 对候选框中提取出的特征，使用分类器判别是否属于一个特定类 5. 对于属于某一特征的候选框，用回归器进一步调整其位置

Faster RCNN 1. 对整张图片输进CNN，得到feature map 2. 卷积特征输入到RPN，得到候选框的特征信息 3. 对候选框中提取出的特征，使用分类器判别是否属于一个特定类 4. 对于属于某一特征的候选框，用回归器进一步调整其位置

*总的来说，从**R-CNN, SPP-NET, Fast R-CNN, Faster R-CNN**一路走来，基于深度学习目标检测的流程变得越来越精简，精度越来越高，速度也越来越快。可以说基于**region proposal**的**R-CNN**系列目标检测方法是当前目标检测技术领域最主要的一个分支。*

R-FCN（2016.5）《R-FCN: Object Detection via Region-based Fully Convolutional Networks》

顾名思议：全卷积网络，就是全部是卷积层，而没有全连接层(fc)。

R-FCN(基于区域的检测器）的方法是：在整个图像上共享计算，通过移除最后的fc层实现(即删除了所有的子网络)。使用“位置敏感的得分图”来解决了图像分类平移不变性与对象检测平移变化之间的矛盾。

此矛盾为：物体分类要求平移不变性越大越好 (图像中物体的移动不用区分)，而物体检测要求有平移变化。所以，ImageNet 分类领先的结果证明尽可能有平移不变性的全卷积结构更受亲睐。另一方面，物体检测任务需要一些平移变化的定位表示。比如，物体的平移应该使网络产生响应，这些响应对描述候选框覆盖真实物体的好坏是有意义的。我们假设图像分类网络的卷积层越深，则该网络对平移越不敏感。

CNN随着网络深度的增加，网络对于位置（Position）的敏感度越来越低，也就是所谓的translation-invariance，但是在Detection的时候，需要对位置信息有很强的的敏感度。

那么ResNet-101的detection是怎么做的?

在R-FCN之前，很简单，把ROI-pooling层放到了前面的卷积层，然后后面的卷积层不共享计算，这样一可以避免过多的信息损失，二可以用后来的卷积层学习位置信息

R-FCN：采用全卷积网络结构作为 FCN，为给 FCN 引入平移变化，用专门的卷积层构建位置敏感分数地图(position-sensitive score maps)。每个空间敏感地图编码感兴趣区域的相对空间位置信息。 在FCN上面增加1个位置敏感 RoI 池化层来监管这些分数地图。

R-FCN思路就是利用最后一层网络通过FCN构成一个position-sensitive的feature map。具体而言，每一个proposal的位置信息都需要编码，那么先把proposal分成kk*个grid，然后对每一个grid进行编码。在最后一层map之后，再使用卷积计算产生一个*kk*(C+1)的map（k*k代表总共的grid数目，C代表class num，+1代表加入一个背景类）

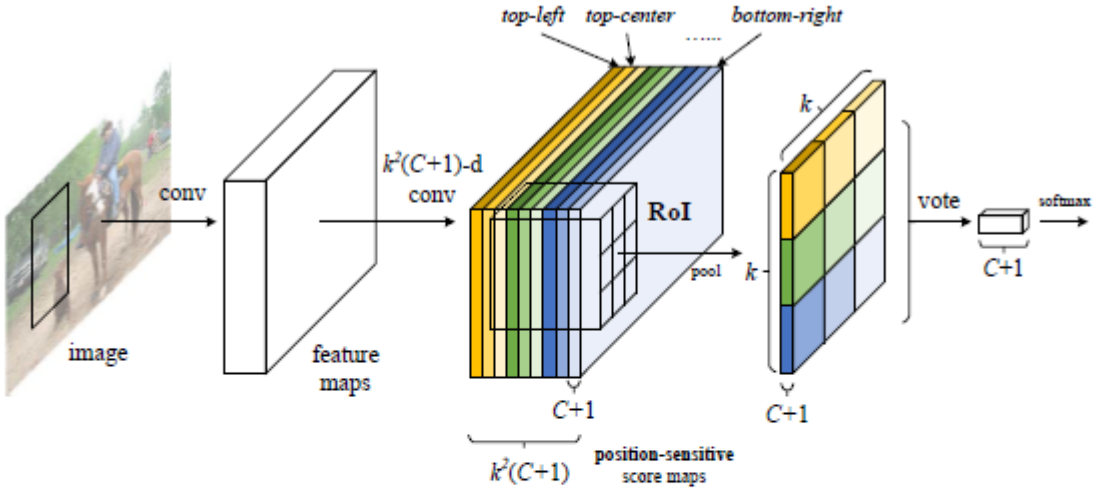


Figure 1: Key idea of **R-FCN** for object detection. In this illustration, there are $k \times k = 3 \times 3$ position-sensitive score maps generated by a fully convolutional network. For each of the $k \times k$ bins in an RoI, pooling is only performed on one of the k^2 maps (marked by different colors).

Table 1: Methodologies of *region-based* detectors using **ResNet-101** [9].

	R-CNN [7]	Faster R-CNN [19, 9]	R-FCN [ours]
depth of shared convolutional subnetwork	0	91	101
depth of RoI-wise subnetwork	101	10	**0**

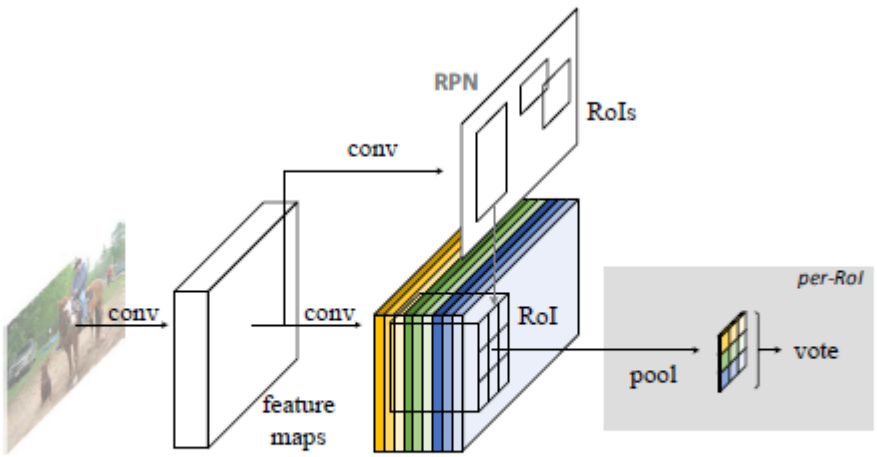


Figure 2: Overall architecture of R-FCN. A Region Proposal Network (RPN) [18] proposes candidate RoIs, which are then applied on the score maps. All learnable weight layers are convolutional and are computed on the entire image; the per-RoI computational cost is negligible.

RPN 给出感兴趣区域，R-FCN 对该感兴趣区域分类。R-FCN 在与 RPN 共享的卷积层后多加1个卷积层。所以，R-FCN 与 RPN 一样，输入为整幅图像。但 R-FCN 最后1个卷积层的输出从整幅图像的卷积响应图像中分

割出感兴趣区域的卷积响应图像。

R-FCN 最后1个卷积层在整幅图像上为每类生成k*k个位置敏感分数图，有C类物体外加1个背景，因此有*k*k(C+1)个通道的输出层。k*k个分数图对应描述位置的空间网格。比如，k×k=3×3，则9个分数图编码单个物体类的 {top−left,top−center,top−right,...,bottom−right}。

R-FCN 最后用位置敏感 RoI 池化层，给每个 RoI 1个分数。选择性池化图解：看上图的橙色响应图像 (top−left)，抠出橙色方块 RoI，池化橙色方块 RoI 得到橙色小方块 (分数)；其它颜色的响应图像同理。对所有颜色的小方块投票 (或池化) 得到1类的响应结果。

产生完了这张map之后，再根据proposal产生一个长宽各为k，channel数目为C+1的score map。具体产生score map的方法是，假如k=3，C=20，那么score map的20个类每个类都有33的feature，一共9个格子，每一个格子都记录了空间信息。而这每一个类的每一个格子都对应前面那个channel数为33*21的大map的其中一个channel的map。现在把score map中的格子对应的区域的map中的信息取平均，然后这个平均值就是score map格子中的值。最后把score map的值进行vote（avg pooling）来形成一个21维的向量来做分类即可。

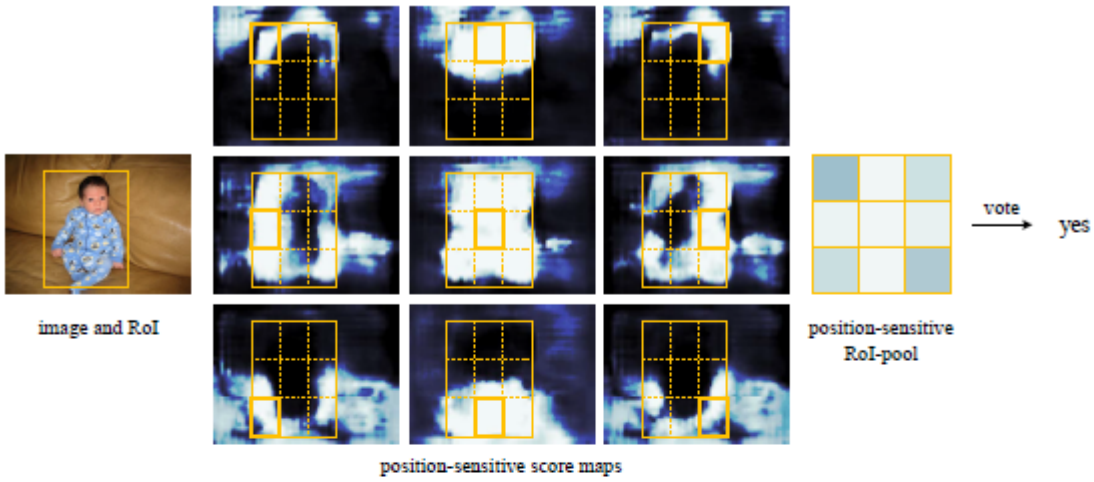


Figure 3: Visualization of R-FCN ($k \times k = 3 \times 3$) for the *person* category.

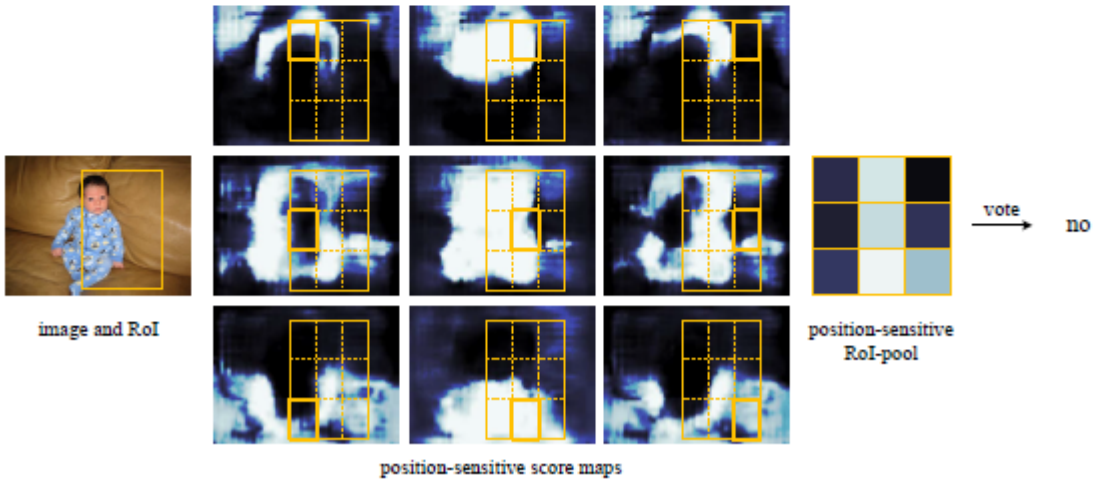


Figure 4: Visualization when an RoI does not correctly overlap the object.

当分类正确时，该类通道的位置敏感分数图 (中间) 的大多数橙色实线网格内的响应在整个 RoI 位置范围内最强。

对应的bbox regression只需要把C+1设成4就可以了。 R-FCN采用的一些方法比Faster R-CNN的baseline提高了3个点，并且比原来Faster R-CNN更快（因为全部计算都共享了）。但是和改进过的Faster R-CNN相比

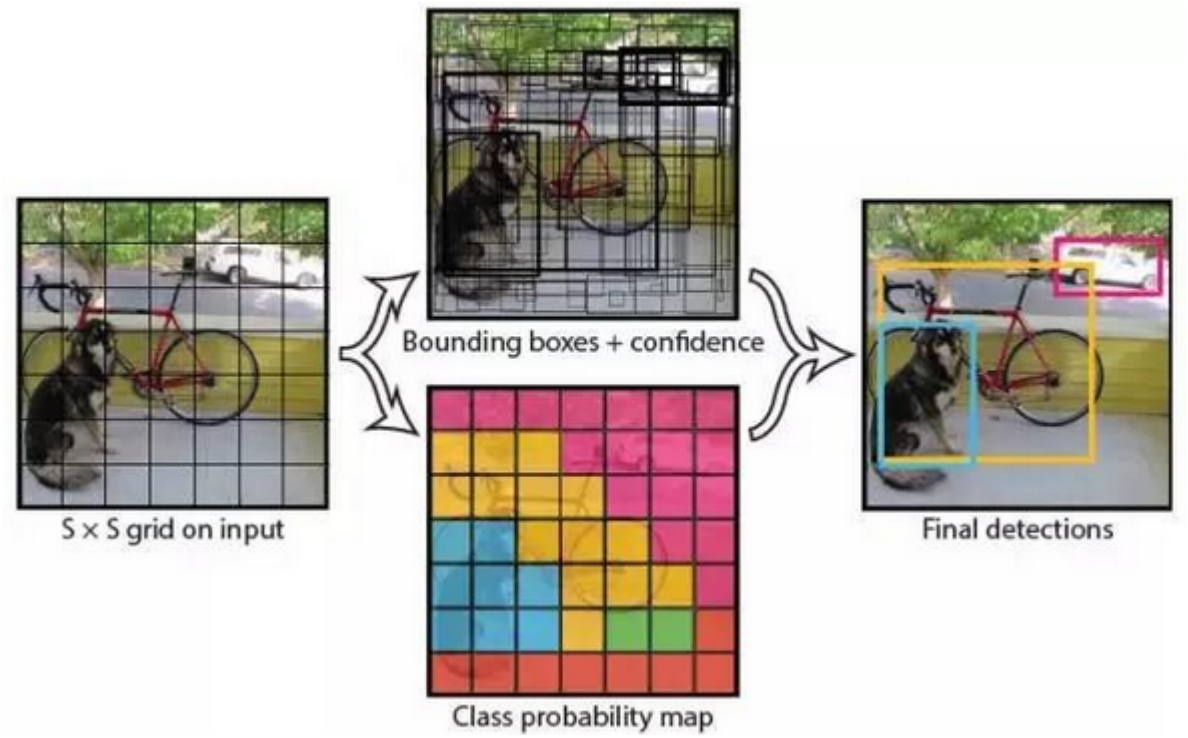
（ROI Pooling提前那种）提高了0.2个点，速度快了2.5倍。所以目前为止这个方法的结果应该是所有方法中速度和Performance结合的最好的。

3.4 基于回归方法的深度学习目标检测算法

Faster R-CNN的方法目前是主流的目标检测方法，但是速度上并不能满足实时的要求。YOLO一类的方法慢慢显现出其重要性，这类方法使用了回归的思想，即给定输入图像，直接在图像的多个位置上回归出这个位置的目标边框以及目标类别。

YOLO (CVPR2016, oral)

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



我们直接看上面YOLO的目标检测的流程图： (1) 给个一个输入图像，首先将图像划分成77*(设S=7)的网格 (2) 对于每个网格，我们都预测2个边框（包括每个边框是目标的置信度以及每个边框区域在多个类别上的概率） (3) 根据上一步可以预测出77*2个目标窗口，然后根据阈值去除可能性比较低的目标窗口，最后NMS去除冗余窗口即可。 可以看到整个过程非常简单，不需要中间的Region Proposal在找目标，直接回归便完成了位置和类别的判定。

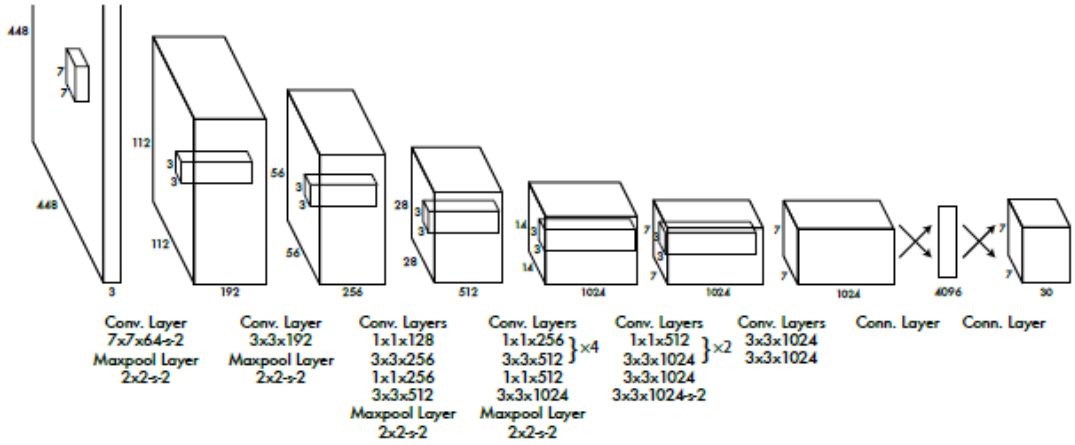


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

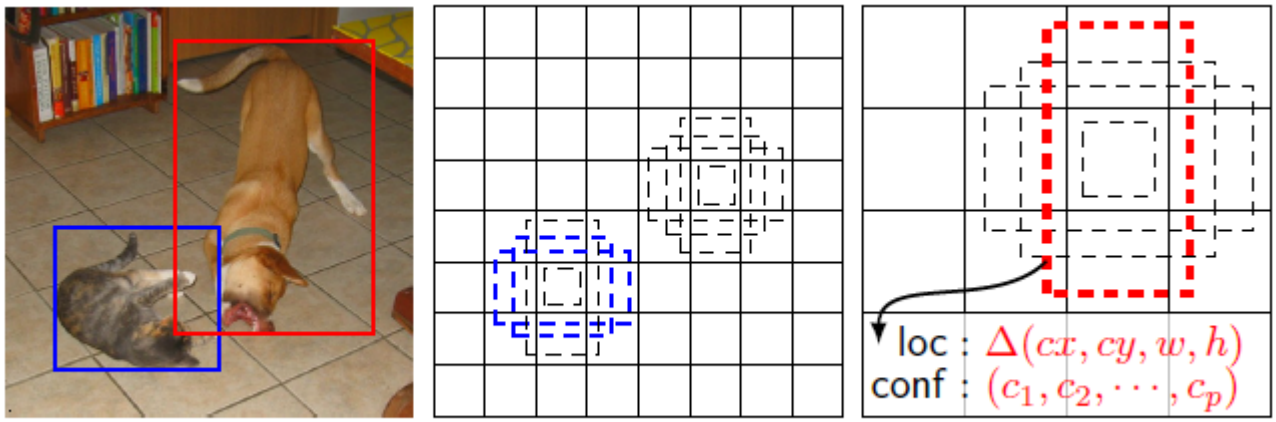
那么如何才能做到直接在不同位置的网格上回归出目标的位置和类别信息呢？上面是YOLO的网络结构图，前边的网络结构跟GoogLeNet的模型比较类似，主要的是最后两层的结构，卷积层之后接了一个4096维的全连接层，然后后边又全连接到一个7730维的张量上。实际上这7*7就是划分的网格数，现在要在每个网格上预测目标两个可能的位置以及这个位置的目标置信度和类别，也就是每个网格预测两个目标，每个目标的信息有4维坐标信息(中心点坐标+长宽)，1个是目标的置信度，还有类别数20(VOC上20个类别)，总共就是(4+1)*2+20 = 30维的向量。这样可以利用前边4096维的全图特征直接在每个网格上回归出目标检测需要的信息（边框信息加类别）。

总结： YOLO将目标检测任务转换成一个回归问题，大大加快了检测的速度，使得YOLO可以每秒处理45张图像。而且由于每个网络预测目标窗口时使用的是全图信息，使得false positive比例大幅降低（充分的上下文信息）。但是YOLO也存在问题：没有了Region Proposal机制，只使用7*7的网格回归会使得目标不能非常精准的定位，这也导致了YOLO的检测精度并不是很高。

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18

SSD(单次检测)

SSD: Single Shot MultiBox Detector 上面分析了YOLO存在的问题，使用整图特征在7*7的粗糙网格内回归对目标的定位并不是很精准。那是不是可以结合Region Proposal的思想实现精准一些的定位？SSD结合YOLO的回归思想以及Faster R-CNN的anchor机制做到了这点。



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

上图是SSD的一个框架图，首先SSD获取目标位置和类别的方法跟YOLO一样，都是使用回归，但是YOLO预测某个位置使用的是全图的特征，SSD预测某个位置使用的是这个位置周围的特征（感觉更合理一些）。那么如何建立某个位置和其特征的对应关系呢？可能你已经想到了，使用Faster R-CNN的anchor机制。如SSD的框架图所示，假如某一层特征图(图b)大小是8*8，*那么就使用*3*3的滑窗提取每个位置的特征，然后这个特征回归得到目标的坐标信息和类别信息(图c)。 不同于Faster R-CNN，这个anchor是在多个feature map上，这样可以利用多层的特征并且自然的达到多尺度（不同层的feature map 3*3滑窗感受野不同）。

小结： SSD结合了YOLO中的回归思想和Faster R-CNN中的anchor机制，使用全图各个位置的多尺度区域特征进行回归，既保持了YOLO速度快的特性，也保证了窗口预测的跟Faster R-CNN一样比较精准。SSD在VOC2007上mAP可以达到72.1%，速度在GPU上达到58帧每秒。

Method	*mAP*	FPS	# Boxes
Faster R-CNN [2](VGG16)	73.2	7	300
Faster R-CNN [2](ZF)	62.1	17	300
YOLO [5]	63.4	45	98
Fast YOLO [5]	52.7	155	98
SSD300	72.1	58	7308
SSD500	**75.1**	23	20097

*总结：**YOLO**的提出给目标检测一个新的思路，**SSD**的性能则让我们看到了目标检测在实际应用中真正的可能性。*

3.5 基于残差（Residual）方法的深度学习目标检测算法

深度残差网络（**Deep Residual Networks**）

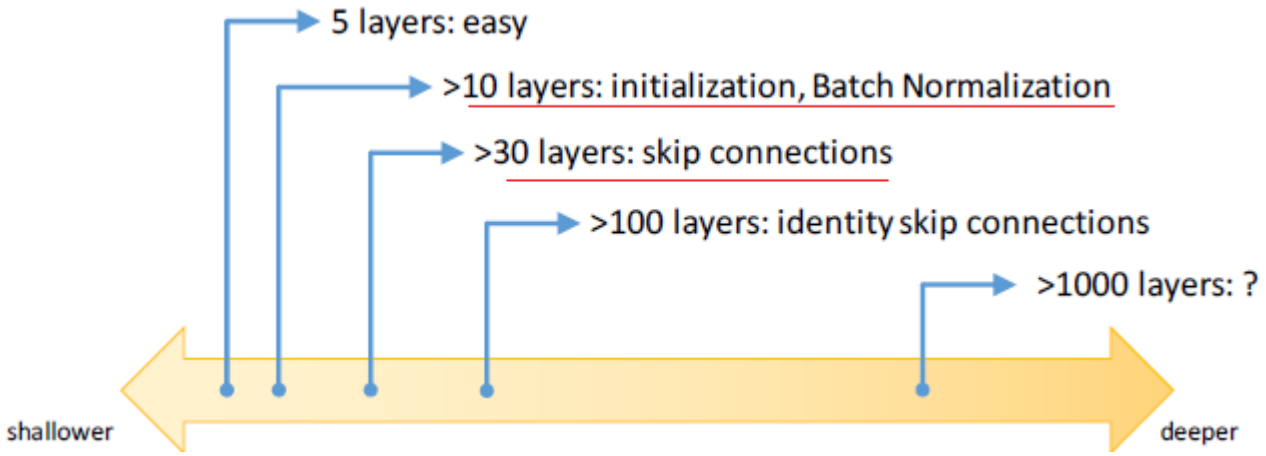
Deep Residual Networks

它使用残差学习的这一思想使得学习更深的网络成为可能，从而学习到更好的表达。

层数越多的神经网络越难以训练。当层数超过一定数量后，传统的深度网络就会因优化问题而出现欠拟合（underfitting）的情况。残差学习框架大幅降低训练更深层网络的难度，也使准确率得到显著提升。在 ImageNet 和 COCO 2015 竞赛中，共有 152 层的深度残差网络 ResNet 在图像分类、目标检测和语义分割各个分项都取得最好成绩，相关论文更是连续两次获得 CVPR 最佳论文。

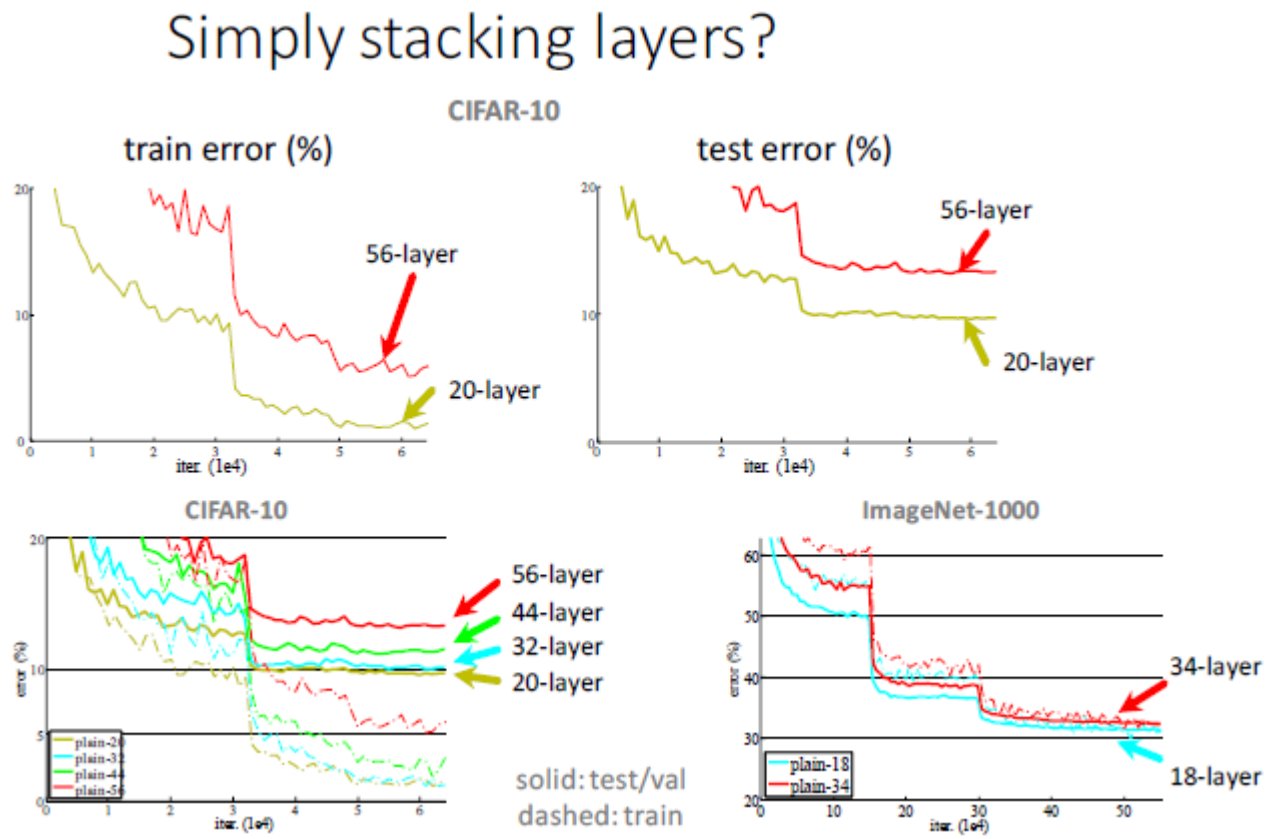
最新研究发现，当残差网络将身份映射作为 skip connection 并实现 inter-block activation，正向和反向信号能够直接从一个区块传播到另一个区块，这样就达到了 1001 层的残差网络。由此可见，神经网络的深度这一非常重要的因素，还有很大的提升空间。

1）深度谱

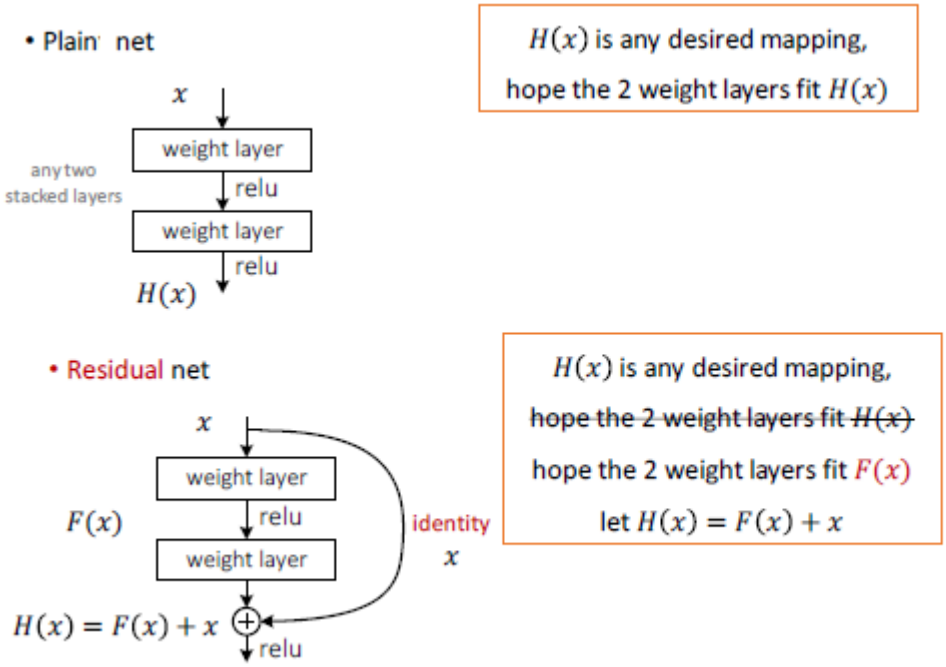


2）为使用网络层数更多，通常采用的方法有：初始化算法，BN方法 3）是否简单堆叠的网络层数越多，训练误差和测试误差就越小?答案正好相反 4）目前流行的深度神经网络结构大致可以分为三类： - 直线型（如AlexNet, VGGNet） - 局部双分支型（ResNet） - 局部多分支型（GoogleNet）

很久以前人们就已经认识到更深的网络能够产生更好的数据表达，但是如何训练一个很深的网络却一直是一个困扰人们的问题，这主要是由于梯度消失或爆炸以及尺度不均匀的初始化造成的。围绕这一问题，人们提出了ReLU、Xavier、pReLU、batch normalization和path-SGD等一系列方法，但是本文作者却发现即使有这些方法，神经网络的训练仍然呈现了degradation的现象。所谓degradation现象，就是随着网络深度的增加，网络的性能反而下降，而且这种性能的下降并不是由前面所说的问题造成的。



4）深度残差学习（Deep Residual Learning）的思想 假如目前有一个可以工作的很好的网络A，这时来了一个比它更深的网络B，只需要让B的前一部分与A完全相同，后一部分只实现一个恒等映射（identity mapping），这样B最起码能获得与A相同的性能，而不至于更差。深度残差学习的思想也由此而产生，既然B后面的部分完成的是恒等映射，何不在训练网络的时候加上这一先验（在网络训练过程中，加入先验信息指导非常重要，合理的先验往往会取得非常好的效果），于是构造网络的时候加入了捷径（shortcut）连接，即每层的输出不是传统神经网络中的输入的映射，而是输入的映射和输入的叠加，如下图中的"Residual net"所示。



在Residual net中： （1）identity：为恒等映射，此条路径一直存在 （2）F(x)：为需要学习的残差函数(residual function)：H(x)-x = F(x)
问题的重新表示或预处理会简化问题的优化! 假设我们期望的网络层关系映射为 H(x), 我们让 the stacked nonlinear layers 拟合另一个映射， F(x):= H(x)-x , 那么原先的映射就是 F(x)+x。 这里我们假设优化残差映射 F(x) 比优化原来的映射 H(x)容易。

这里我们首先求取残差映射 F(x):= H(x)-x，那么原先的映射就是 F(x)+x。尽管这两个映射应该都可以近似理论真值映射 the desired functions (as hypothesized)，但是它俩的学习难度是不一样的。

这种改写启发于"网络层数越多，训练和测试误差越大"性能退化问题违反直觉的现象。如果增加的层数可以构建为一个恒等映射(identity mappings)，那么增加层数后的网络训练误差应该不会增加，与没增加之前相比较。性能退化问题暗示多个非线性网络层用于近似identity mappings 可能有困难。使用残差学习改写问题之后，如果identity mappings 是最优的，那么优化问题变得很简单，直接将多层非线性网络参数趋0。

实际中，identity mappings 不太可能是最优的，但是上述改写问题可能帮助预处理问题。如果最优函数接近 identity mappings，那么优化将会变得容易些。 实验证明该思路是对的。

F(x)+x 可以通过shortcut connections 来实现，如下图所示：

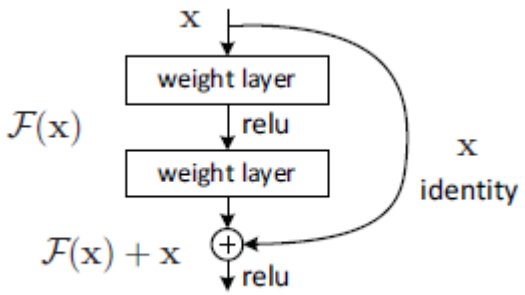


Figure 2. Residual learning: a building block.

.....And so on (see reference)

相关文献

1. 如何选择物体检测器——对当下主流CNN物体检测器的评测 2. facebookresearch/Detectron 各种网络实现 3. A paper list of object detection using deep learning. I worte this page with reference to this survey paper and searching and searching 4. RCNN, Fast R-CNN 与 Faster RCNN理解及改进方法 YOLO v1到v3的进化之路 5. 人工智能从业者应该从哪里去寻找和阅读顶级论文？ 6. 计算机视觉入门系列（一） 综述 ***7.【更新中】目标检测——梳理，准备面试***