目标检测算法

目标检测是很多计算机视觉任务的基础，不论我们需要实现图像与文字的交互还是需要识别精细类别，它都提供了可靠的信息。

第一部分从 RCNN 开始介绍基于候选区域的目标检测，包括 Fast R-CNN、Faster R-CNN 和 R-FCN 等；

第二部分则重点讨论了包括 YOLO、SSD 和 RetinaNet 等在内的单次检测器，它们都是目前最为优秀的方法。

目标检测目前有 one-stage 和 two-stage 两种，two-stage 指的是检测算法需要分两步完成，首先需要获取候选区域（RPN），然后进行分类，比如R-CNN系列；与之相对的是 one-stage 检测，可以理解为一步到位，不需要单独寻找候选区域，典型的有SSD/YOLO。

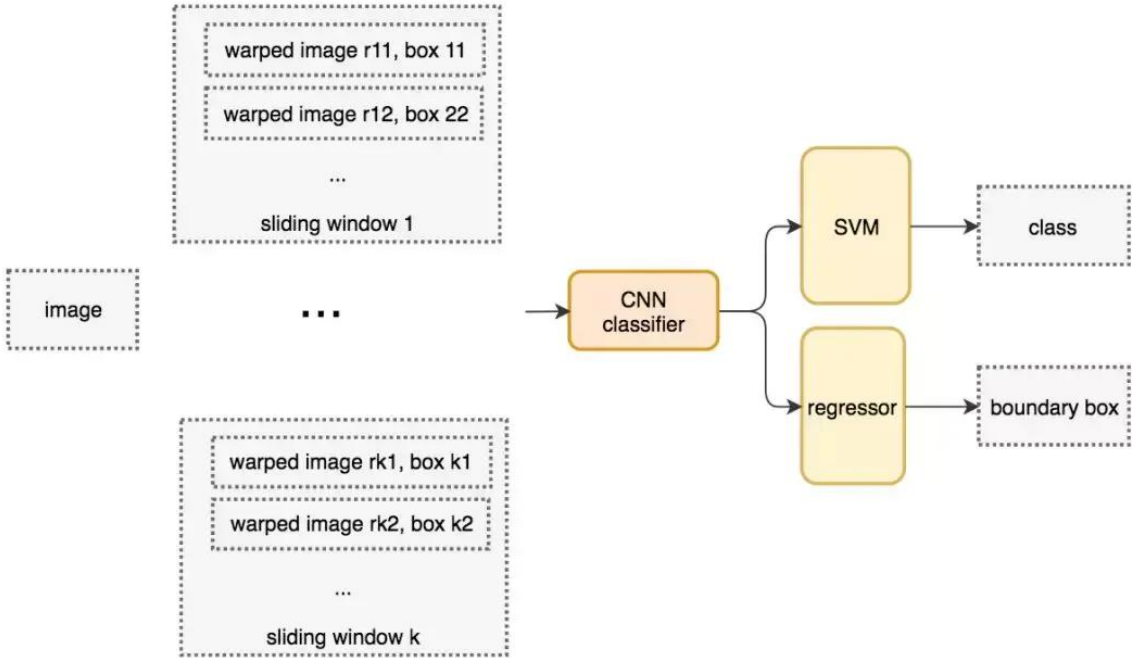
two_stage：1. 滑动窗口检测器

自从 AlexNet 获得 ILSVRC 2012 挑战赛冠军后，用 CNN 进行分类成为主流。一种用于目标检测的暴力方法是从左到右、从上到下滑动窗口，利用分类识别目标。为了在不同观察距离处检测不同的目标类型，需要使用不同大小和宽高比的窗口。

根据滑动窗口从图像中剪切图像块。由于很多分类器只取固定大小的图像，因此这些图像块是经过变形转换的。但是，这不影响分类准确率，因为分类器可以处理变形后的图像。

two_stage：1. 滑动窗口检测器

变形图像块被输入 CNN 分类器中，提取出 4096 个特征。之后，我们使用 SVM 分类器识别类别和该边界框的另一个线性回归器。



下面是伪代码。我们创建很多窗口来检测不同位置的不同目标，要提升性能，一个显而易见的办法就是减少窗口数量。

```
for window in windows:
    patchs = get_patch(image, window)
    results = detector(patchs)
```

two_stage: 2. R-CNN

实际上，更实用的方法是候选区域（Region Proposals ）方法来获取感兴趣的区域（ROI）。

选择性搜索（Selective Search ）就是一种典型的候选区域方法。

首先通过简单的区域划分算法，将图片划分成很多小区域，再通过相似度和区域大小（小的区域先聚合，这样是防止大的区域不断的聚合小区域，导致层次关系不完全）不断的聚合相邻小区域，类似于聚类的思路。这样就能解决object层次问题。

算法原理如下：首先将每个像素作为一组。然后，计算每一组的纹理，并将两个最接近的组结合起来。但是为了避免单个区域吞噬其他区域，首先对较小的组进行分组。继续合并区域，直到所有区域都结合在一起。下图第一行展示了如何使区域增长，第二行中的蓝色矩形代表合并过程中所有可能的 ROI。最后对于计算速度来说，只能说这个思路在相对穷举，大大减少了后期分类过程中的计算量。

two_stage: 2. R-CNN

区域合并采用了多样性的策略，如果简单采用一种策略很容易错误合并不相似的区域，比如只考虑纹理时，不同颜色的区域很容易被误合并。选择性搜索采用三种多样性策略来增加候选区域以保证召回：

多种颜色空间，考虑RGB、灰度、HSV及其变种等
多种相似度度量标准，既考虑颜色相似度，又考虑纹理、大小、重叠情况等。
通过改变阈值初始化原始区域，阈值越大，分割的区域越少。

colour spaces	RGB	I	Lab	rgI	HSV	rgb	C	H
Light Intensity	-	-	+/-	2/3	2/3	+	+	+
Shadows/shading	-	-	+/-	2/3	2/3	+	+	+
Highlights	-	-	-	-	1/3	-	+/-	+

为了保证能够划分的完全，对于相似度，作者提出了可以多样化的思路，不但使用多样的颜色空间（（1）RGB，（2）灰度I，（3）Lab，（4）rgI（归一化的rg通道加上灰度），（5）HSV，（6）rgb（归一化的RGB），（7）C，（8）H（HSV的H通道）），还有很多不同的相似度计算方法。
然后通过多样性的距离计算方式，综合颜色、纹理等所有的特征。

two_stage: 2. R-CNN

我们在计算多种相似度的时候，都是把单一相似度的值归一化到[0,1]之间，1表示两个区域之间相似度最大。

- **颜色相似度**

  使用L1-norm归一化获取图像每个颜色通道的25 bins的直方图，这样每个区域都可以得到一个75维的向量 $\{c_i^1, \cdots, c_i^n\}$，区域之间颜色相似度通过下面的公式计算：

$$s_{colour}(r_i, r_j) = \sum_{k=1}^{n} \min(c_i^k, c_j^k).$$

  上面这个公式可能你第一眼看过去看不懂，那咱们打个比方，由于 $\{c_i^1, \cdots, c_i^n\}$ 是归一化后值，每一个颜色通道的直方图累加和为1.0，三个通道的累加和就为3.0，如果区域c_i和区域c_j直方图完全一样，则此时颜色相似度最大为3.0，如果不一样，由于累加取两个区域bin的最小值进行累加，当直方图差距越大，累加的和就会越小，即颜色相似度越小。

  在区域合并过程中使用需要对新的区域进行计算其直方图，计算方法：

$$C_t = \frac{\text{size}(r_i) \times C_i + \text{size}(r_j) \times C_j}{\text{size}(r_i) + \text{size}(\text{r}_\text{j})}.$$

two_stage: 2. R-CNN

- 纹理相似度

这里的纹理采用SIFT-Like特征。具体做法是对每个颜色通道的8个不同方向计算方差σ=1的高斯微分（Gaussian Derivative），使用L1-norm归一化获取图像每个颜色通道的每个方向的10 bins的直方图，这样就可以获取到一个240（10x8x3）维的向量 $T_i = \{t_i^1, \cdots, t_i^n\}$，区域之间纹理相似度计算方式和颜色相似度计算方式类似，合并之后新区域的纹理特征计算方式和颜色特征计算相同：

$$s_{texture}(r_i, r_j) = \sum_{k=1}^{n} \min(t_i^k, t_j^k).$$

- 优先合并小的区域

如果仅仅是通过颜色和纹理特征合并的话，很容易使得合并后的区域不断吞并周围的区域，后果就是多尺度只应用在了那个局部，而不是全局的多尺度。因此我们给小的区域更多的权重，这样保证在图像每个位置都是多尺度的在合并。

$$s_{size}(r_i, r_j) = 1 - \frac{\text{size}(r_i) + \text{size}(\text{r}_\text{j})}{\text{size}(im)},$$

上面的公式表示，两个区域越小，其相似度越大，越接近1。

two_stage: 2. R-CNN

- 区域的合适度距离

如果区域r_i包含在r_j内，我们首先应该合并，另一方面，如果r_i很难与r_j相接，他们之间会形成断崖，不应该合并在一块。这里定义区域的合适度距离主要是为了衡量两个区域是否更加“吻合”，其指标是合并后的区域的Bounding Box（能够框住区域的最小矩形BB_{ij}）越小，其吻合度越高，即相似度越接近1。其计算方式：

$$fill(r_i, r_j) = 1 - \frac{\text{size}(BB_{ij}) - \text{size}(r_i) - \text{size}(r_i)}{\text{size}(im)}$$

- 合并上面四种相似度

$$\begin{aligned} s(r_i, r_j) = \; & a_1 s_{colour}(r_i, r_j) + a_2 s_{texture}(r_i, r_j) + \\ & a_3 s_{size}(r_i, r_j) + a_4 s_{fill}(r_i, r_j), \end{aligned}$$

其中$a_i \in \{0, 1\}$

给区域打分

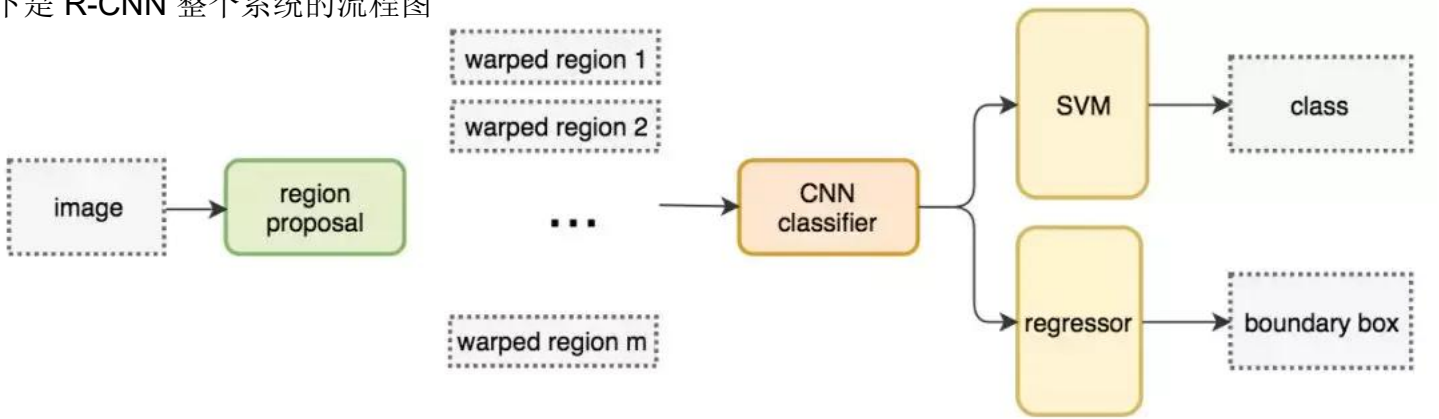
通过上述的步骤我们能够得到很多很多的区域，但是显然不是每个区域作为目标的可能性都是相同的，因此需要衡量这个可能性，这样就可以根据需要筛选区域建议个数（2000）。

这篇文章做法是，给予最先合并的图片块较大的权重，比如最后一块完整图像权重为1，倒数第二次合并的区域权重为2以此类推。但是当策略很多，多样性很多的时候，这个权重就会有太多的重合了，排序有问题。文章做法是乘以一个随机数，对于相同的区域多次出现的也叠加下权重。这样就得到了所有区域的目标分数，也就可以根据自己的需要选择需要多少个区域了。

two_stage: 2. R-CNN

R-CNN 利用候选区域方法创建了约 2000 个 ROI。这些区域被转换为固定大小的图像（暴力转换，Alextnet 的输入图像大小是 227x227），并分别送到卷积神经网络中。之后使用SVM对区域进行分类，使用线性回归损失来校正边界框，以实现目标分类并得到边界框。

以下是 R-CNN 整个系统的流程图



通过使用更少且更高质量的 ROI，R-CNN 要比滑动窗口方法更快速、更准确。伪代码如下：

```
ROIs = region_proposal(image)
for ROI in ROIs
    patch = get_patch(image, ROI)
    results = detector(patch)
```

two_stage: 2. R-CNN

1、输入一张多目标图像，采用selective search算法提取约2000个建议框；

2、先在每个建议框周围加上16个像素值为建议框像素平均值的边框，再直接变形为227×227的大小；

3、先将所有建议框像素减去该建议框像素平均值后【预处理操作】，再依次将每个227×227的建议框输入AlexNet CNN网络获取4096维的特征【比以前的人工经验特征低两个数量级】，2000个建议框的CNN特征组合成2000×4096维矩阵；

4、将2000×4096维特征与20个SVM组成的权值矩阵4096×20相乘【20种分类，SVM是二分类器，则有20个SVM】，获得2000×20维矩阵表示每个建议框是某个物体类别的得分；

5、分别对上述2000×20维矩阵中每一列即每一类进行NMS剔除重叠建议框，得到该列即该类中得分最高的一些建议框；

6、分别用20个回归器对上述20个类别中剩余的建议框进行回归操作，最终得到每个类别的修正后的得分最高的bounding box。

two_stage: 2. R-CNN

因为最终目标分类是通过SVM进行分类的，而不是通过网络框架中的softmax分类的。

下面先说一下在SVM的训练中，正负样本的定义，为什么这样定义，然后再说一下为什么不直接用softmax输出的结果而是再训练SVM来进行分类的。

1）SVM正负样本的定义，为什么fine-tuning与SVM正负样本定义不一样？
在训练SVM时，正样本为groundtruth，负样本定义为与ground truth的IoU小于0.3的候选区域为负样本，介于0.3与0.7之间的样本忽略。
微调期间，定义与ground truth的IoU大于0.5的候选区域为正样本，其余的为负样本，fine-tuning时担心过拟合的原因，要扩大正样本的样本量，所以定义比较宽松，但是SVM是最终用于分类的分类器，而且SVM原理就是最小的距离最大化，越难分的数据越有利于SVM的训练，所以对样本的定义比较严格。

2）为什么不直接用softmax的输出结果？
因为在训练softmax的时候数据本来就不是很准确，而SVM的训练使用的是hard negative也就是样本比较严格，所以SVM效果会更好

SS正负样本不均衡，SVM减少正负样本不均衡的影响。

two_stage: 2. R-CNN

SVM本质模型是特征空间中最大化间隔的线性分类器，是一种二分类模型。

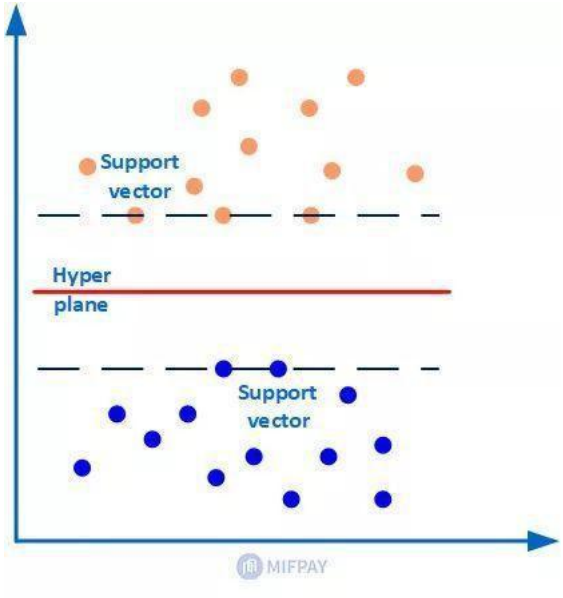
之所以叫支持向量机，因为其核心理念是：支持向量样本会对识别的问题起关键性作用。那什么是支持向量（Support vector）呢？支持向量也就是离分类超平面（Hyper plane）最近的样本点。

只有当样本数据是线性可分的，才能找到一条线性分割线或分割面等，SVM分类才能成立。假如样本特征数据是线性不可分的，则这样的线性分割线或分割面是根本不存在的，SVM分类也就无法实现

分类超平面的确定仅取决于支持向量。
对于给定的训练样本，首先要找到距离分类超平面最近的点（支持向量），再通过最大化这些点之间的间隔来求解。

而如果样本数据是非线性的情况，那将如何处理呢？SVM的解决办法就是先将数据变成线性可分的，再构造出最优分类超平面。

SVM 通过选择一个核函数 K，将低维非线性数据映射到高维空间中。原始空间中的非线性数据经过核函数映射转换后，在高维空间中变成线性可分的数据，从而可以构造出最优分类超平面。



two_stage: 2. R-CNN

简单地说，核函数是计算两个向量在隐式映射后空间中的内积的函数。核函数通过先对特征向量做内积，然后用函数K 进行变换，这有利于避开直接在高维空间中计算，大大简化问题求解。并且这等价于先对向量做核映射然后再做内积。

名称	计算表达式
线性核函数	$K(X1, X2)= \langle X1, X2 \rangle$
多项式核函数	$K(X1, X2)=(\langle X1, X2 \rangle+R)^{d}$
高斯核函数	$K(X1, X2)= \exp(-\|\|X1-X2\|\|^{2}/2\sigma^{2})$

线性核函数，就是简单原始空间中的内积。

多项式核函数，可根据R和d的取值不同，而有不同的计算式。

高斯核函数，可根据实际需要灵活选取参数σ，甚至还可以将原始维度空间映射到无穷维度空间。不过，如果σ取值很大，会导致高次特征上的权重衰减快；如果σ取值很小，其好处是可以将任意的数据映射成为线性可分，但容易造成过拟合现象。
在实际应用中，如果训练样本数量大，经训练后得出的模型中支持向量的数量会有很多，利用该模型进行新样本预测时，需要先计算新样本与每个支持向量的内积，然后做函数 K转换，耗时长、速度慢。

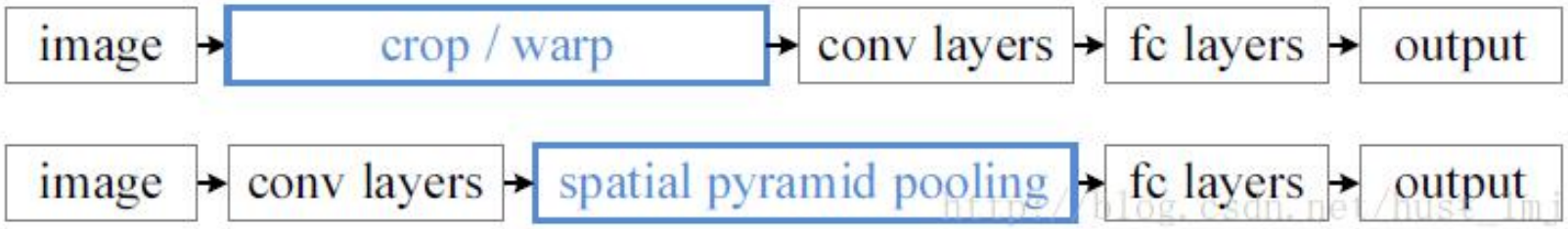
two_stage: 3. SPP

R-CNN中单幅图片大约有2000个region proposal，重复用CNN为每个region proposal提取特征是极其费时的，因此SPP-net提出：能否在feature map上提取ROI（即region proposal）特征，这样就只需要在整幅图像上做一次卷积，大大提高了速度。这样就要求SPP-NET解决两个问题：

1. 原始图像的ROI如何映射到特征图（一系列卷积层的最后输出）

2. ROI在特征图上的对应的特征区域的维度不满足全连接层的输入要求怎么办（又不可能像R-CNN一样在输入CNN之前对原始ROI图像上那样进行wrap）.

事实上，CNN的卷积层不需要固定尺寸的图像，全连接层是需要固定大小输入的，因此提出了SPP层放到卷积层的后面，全连接层的前面改进后的网络如下图所示：



two_stage: 3. SPP

如何使feature map上的ROI对应的维度满足全连接层固定输入维度的要求.

使用空间金字塔池化层(Spatial Pyramid Pooling)替代原来的pool5，思路是对于任意大小的feature map(将原图任意大小的ROI映射到全图卷积后的feature map上得到的不同尺寸的小feature map)，例如首先将其分成16、4、1个块(window size)，然后在每个块上最大池化，池化后的特征拼接得到一个固定维度的输出，以满足全连接层的需要。
假设原图输入是224x224，对于conv5出来后的输出是13x13x256的，。如果像上图那样将reponse map分成1x1，2x2，4x4三张子图，分别做max pooling后，出来的特征就是(16+4+1)x256 维度。如果原图的输入不是224x224，出来的特征依然是(16+4+1)x256维度。这样就实现了不管图像尺寸如何，SPP池化的输出永远是 （16+4+1）x256 维度。实际运用中只需要根据全连接层的输入维度要求设计好空间金字塔即可。

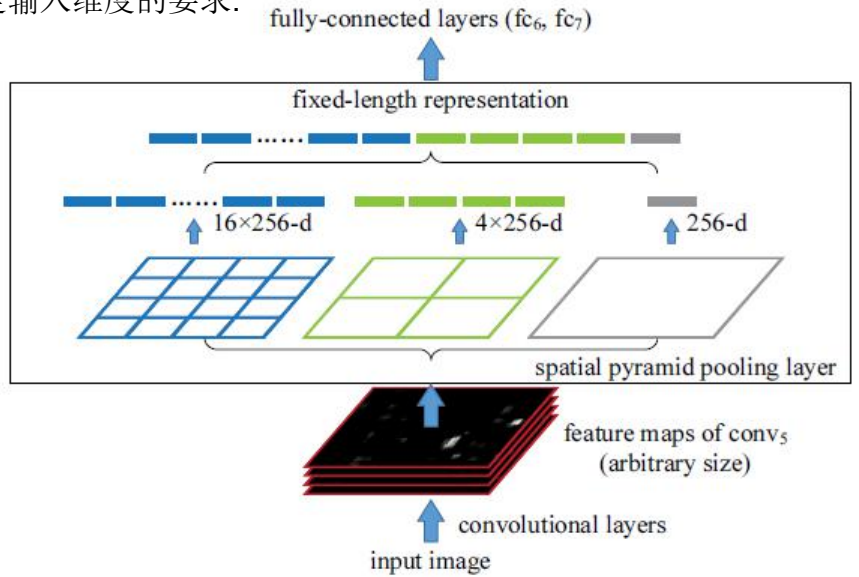
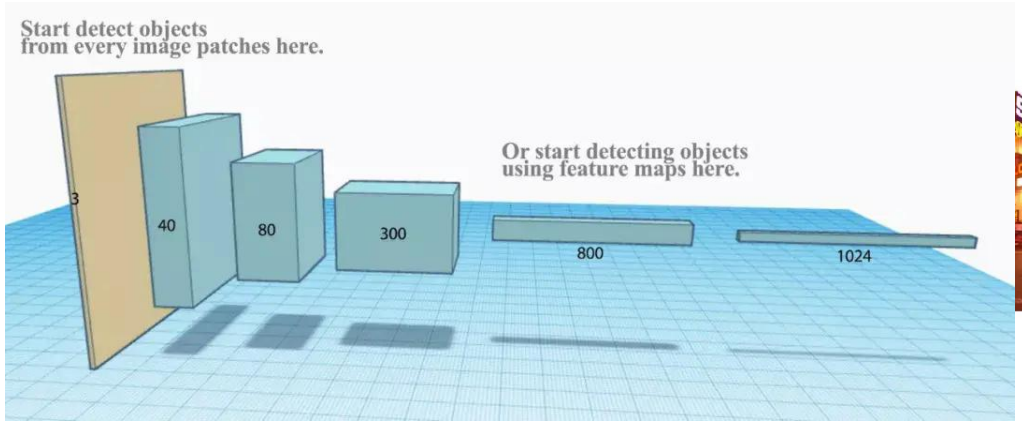


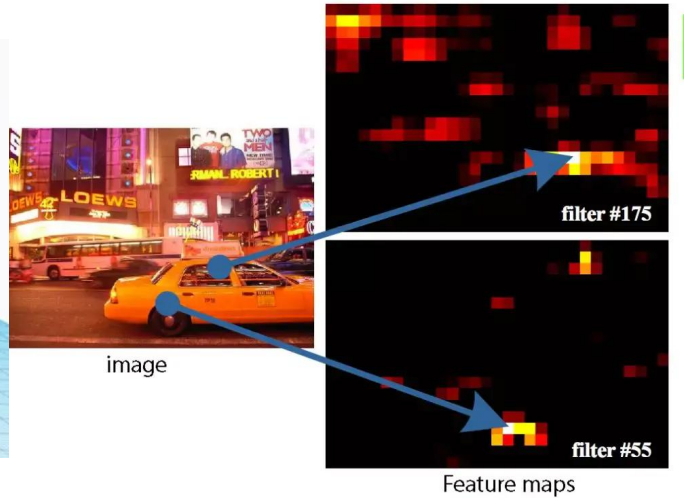
Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the $conv_5$ layer, and $conv_5$ is the last convolutional layer.

two_stage: 4. Fast R-CNN

R-CNN 需要非常多的候选区域以提升准确度，但其实有很多区域是彼此重叠的。如果我们有 2000 个候选区域，且每一个都需要独立地馈送到 CNN 中，那么对于不同的 ROI，我们可能需要重复提取很多次特征。因此 R-CNN 的训练和预测速度非常慢。

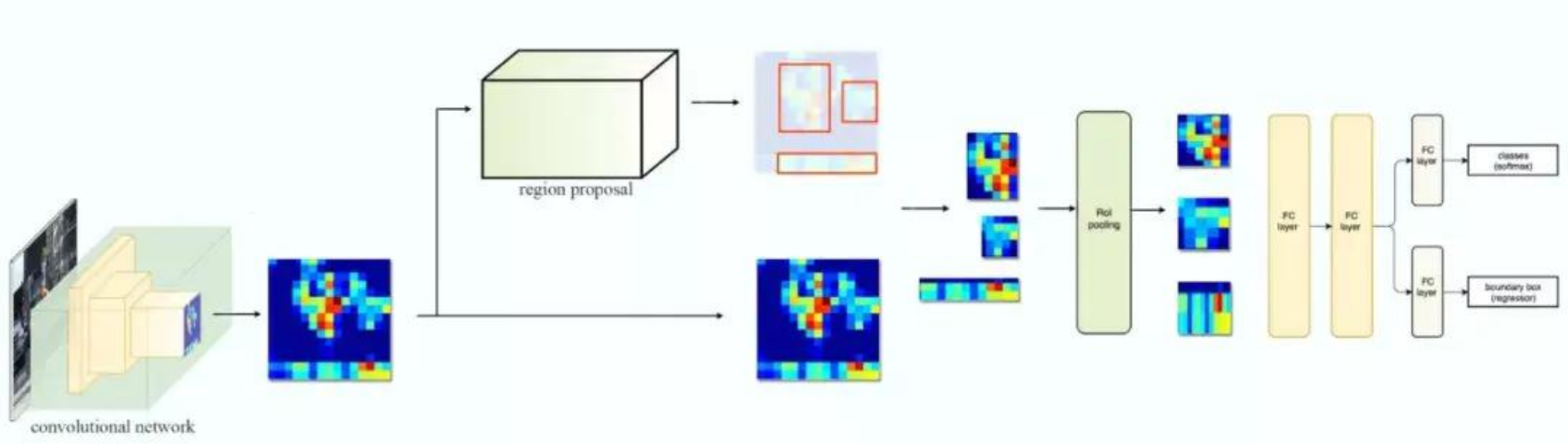
可以直接使用特征图代替原图来检测目标





two_stage: 4. Fast R-CNN

Fast R-CNN 使用CNN网络先提取整个图像的特征，而不是对每个图像块提取多次。然后，我们可以将创建候选区域的方法直接应用到提取到的特征图上。例如，Fast R-CNN 选择了 VGG16 中的卷积层 conv5 来生成 ROI区域在对应的特征图上的映射特征图块，并用于目标检测任务中。我们使用 ROI 池化将特征图块转换为固定的大小，并送到全连接层进行分类和定位。因为 Fast-RCNN 不会重复提取特征，所以它能显著地减少处理时间。



two_stage: 4. Fast R-CNN

ROI 池化

因为 Fast R-CNN 使用全连接层，所以我们应用 ROI 池化将不同大小的 ROI 转换为固定大小。比如我们将 8×8 特征图转换为预定义的 2×2 大小：

左上角：输入特征图；
右上角：将 ROI（蓝色区域）与特征图重叠；
左下角：将 ROI 拆分为目标维度。例如，对于 2×2 目标，我们将 ROI 分割为 4 个大小相似或相等的部分；
右下角：找到每个部分的最大值，得到变换后的特征图。

按上述步骤得到一个 2×2 的特征图块，可以送至分类器和边界框回归器中。使用softmax损失进行分类；使用回归损失比如平滑L1损失校正包围框。总损失是两部分的和，然后反向传播进行训练。Fast R-CNN 最重要的一点就是包含特征提取器、分类器和边界框回归器在内的整个网络能通过多任务损失函数进行端到端的训练，这种多任务损失即结合了分类损失和定位损失的方法，大大提升了模型准确度。

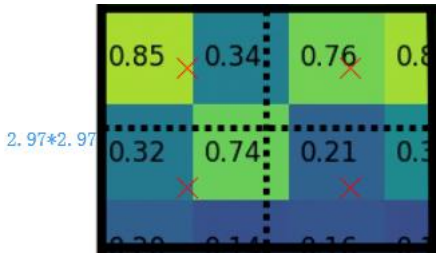
0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

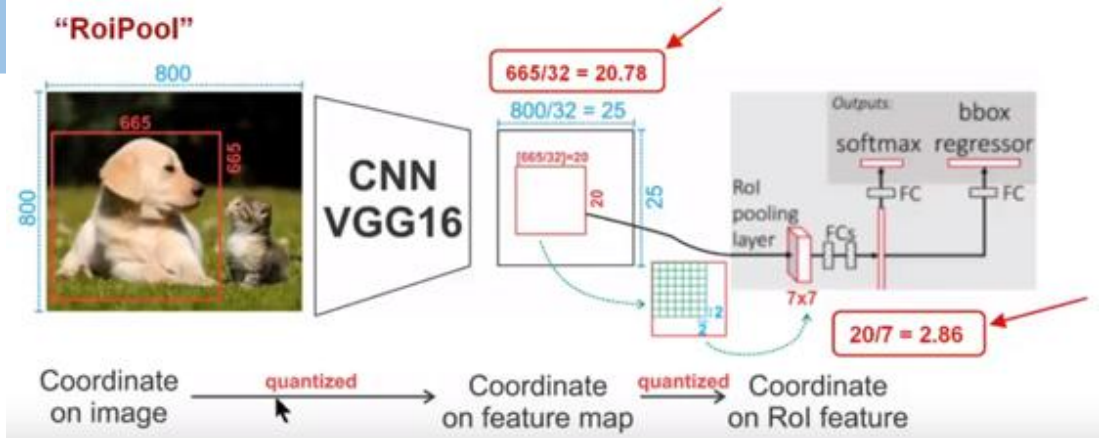
0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

0.8	0.6
0.9	0.6

two_stage: 4. Fast R-CNN





1)Conv layers使用的是VGG16，feat_stride=32(即表示，经过网络层后图片缩小为原图的1/32),原图800*800,最后一层特征图feature map大小:25*25

2)假定原图中有一region proposal，大小为665*665，这样，映射到特征图中的大小：665/32=20.78,即20.78*20.78，此时，没有像RoiPooling那样就行取整操作，保留浮点数

3)假定pooled_w=7,pooled_h=7,即pooling后固定成7*7大小的特征图，所以，将在 feature map上映射的20.78*20.78的region proposal 划分成49个同等大小的小区域，每个小区域的大小20.78/7=2.97,即2.97*2.97

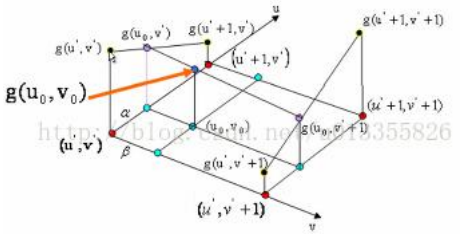
4)假定采样点数为4，即表示，对于每个2.97*2.97的小区域，平分四份，每一份取其中心点位置，而中心点位置的像素，采用双线性插值法（下面介绍）进行计算，这样，就会得到四个点的像素值。

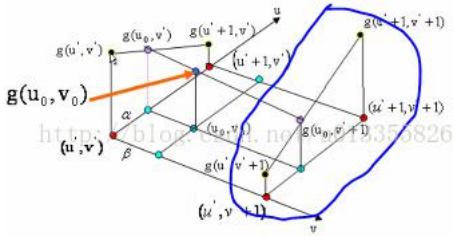
two_stage: 5. Fast R-CNN

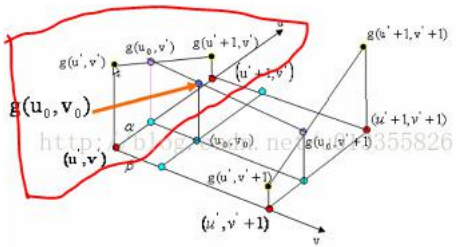
为什么要用双线性插值法

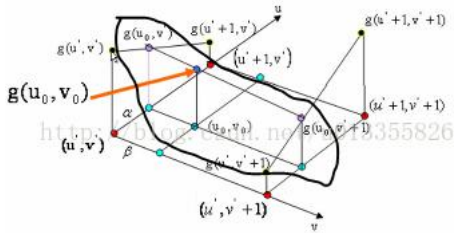
在图像的放大和缩小的过程中，需要计算新图像像素点在原图的位置，如果计算的位置不是整数，就需要用到图像的内插，我们需要寻找在原图中最近得像素点赋值给新的像素点，这种方法很简单是最近邻插法，这种方法好理解、简单，但是不实用，会产生是真现象，产生棋盘格效应，更实用的方法就是双线性插值法。

双线性插值是做了二次一维的线性插值，我们用四个最近邻估计给定的灰度。我们新图像的像素点对应输入图像的（u0 , v0）(u0,v0不是整数)，则其必定落在原始图像四个像素点中间。四个像素点分别是（u' , v' ）、（u' , v' +1）、（u'+1 , v' ）、（u' +1, v'+1 ）。









two_stage: 4. Fast R-CNN

在下面的伪代码中，计算量巨大的特征提取过程从 For 循环中移出来了，因此速度得到显著提升。Fast R-CNN 的训练速度是 R-CNN 的 10 倍，预测速度是后者的 150 倍。

RCNN:

```
ROIs = region_proposal(image)
for ROI in ROIs
    patch = get_patch(image, ROI)
    results = detector(patch)
```

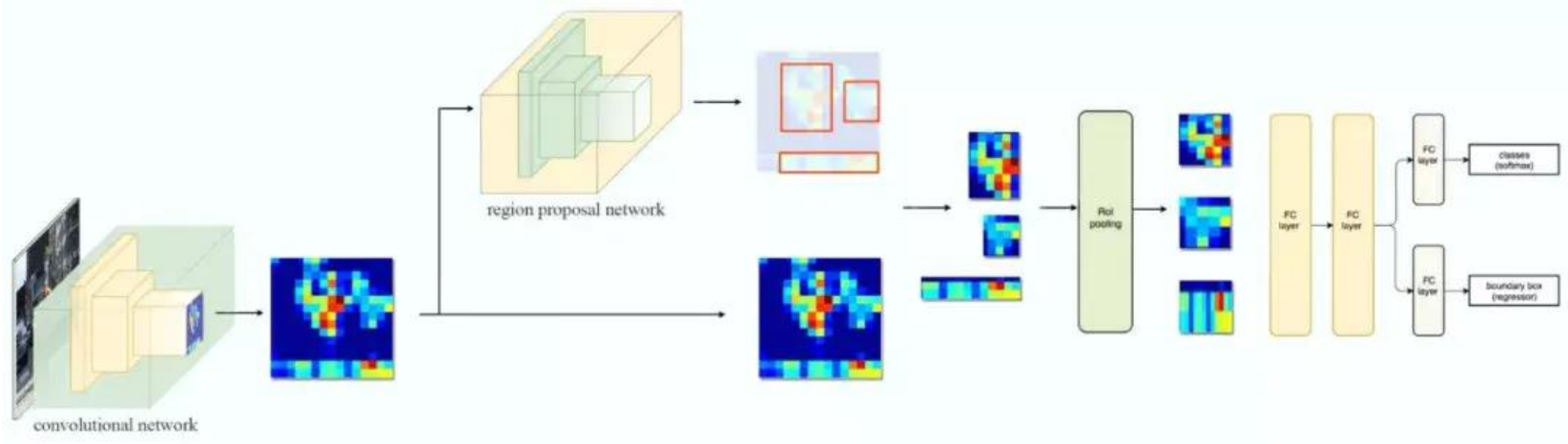
FAST RCNN:

```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
for ROI in ROIs
    patch = roi_pooling(feature_maps, ROI)
    results = detector2(patch)
```

two_stage: 5. Faster R-CNN

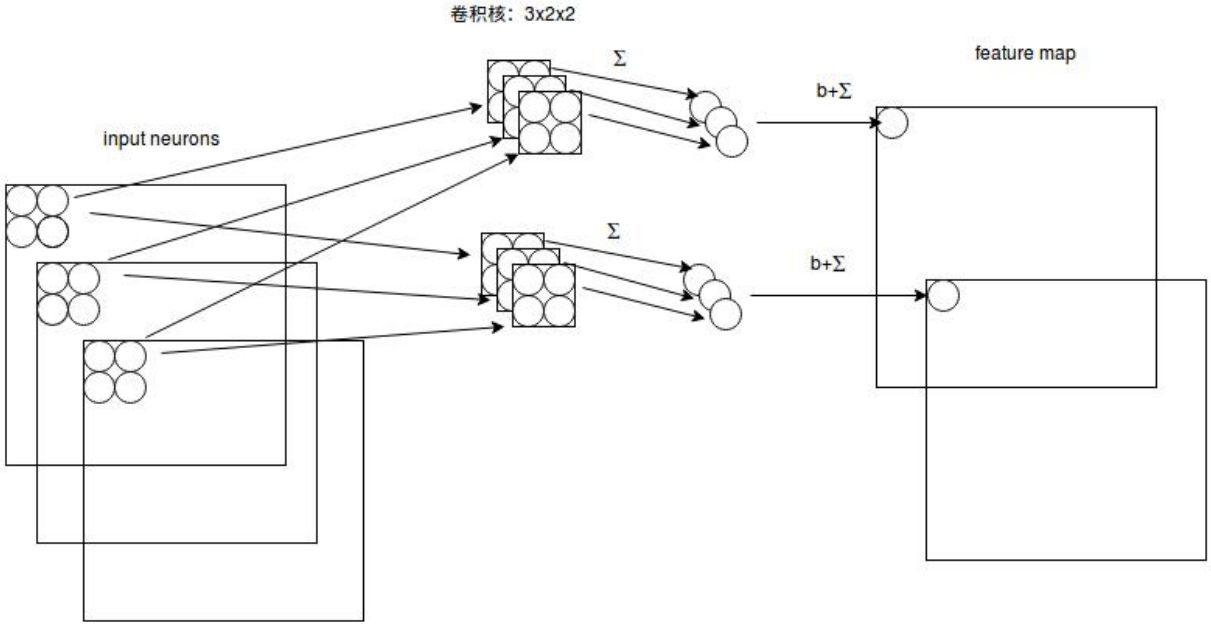
Fast R-CNN 依赖于外部候选区域方法，如选择性搜索。但这些算法在 CPU 上运行且速度很慢。在测试中，Fast R-CNN 需要 2.3 秒来进行预测，其中 2 秒用于生成 2000 个 ROI。因此区域生成的计算成为整个检测网络的瓶颈。

与其使用固定的算法得到候选区域，不如让网络自己学习自己的候选区域应该是什么。因此，Faster R-CNN 采用与 Fast R-CNN 相同的设计，只是它用区域生成网络（Region Proposal Network，RPN）代替了候选区域方法。新的候选区域网络（RPN）在生成 ROI 时效率更高，并且以每幅图像 10 毫秒的速度运行。



two_stage: 5. Faster R-CNN

多通道图像卷积基础知识介绍
对于多通道图像+多卷积核做卷积，计算方式如下：



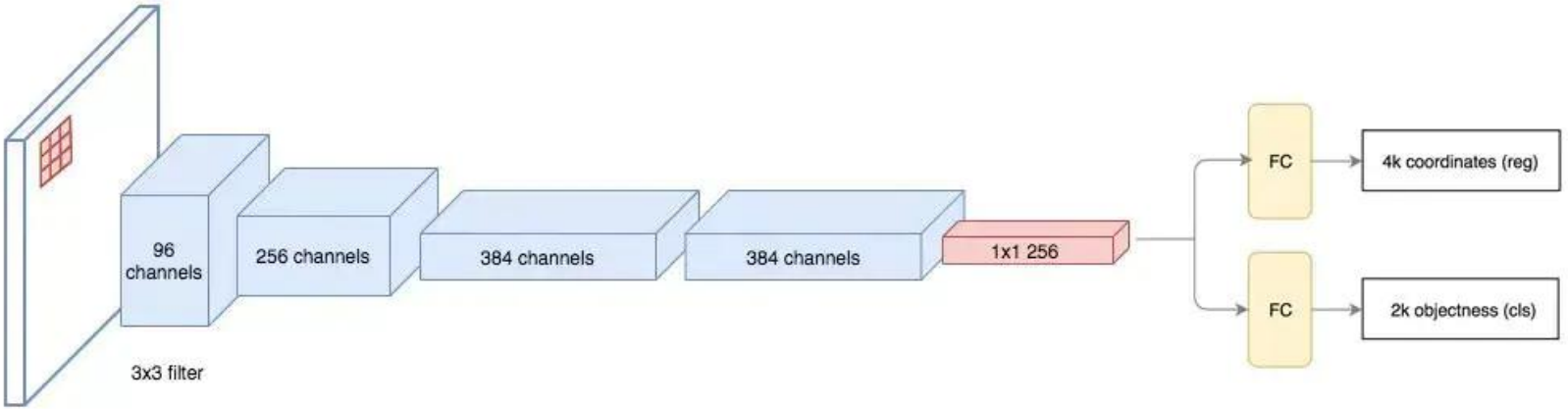
输入有3个通道，同时有2个卷积核。对于每个卷积核，先在输入3个通道分别作卷积，再将3个通道结果加起来得到卷积输出。所以对于某个卷积层，无论输入图像有多少个通道，输出图像通道数总是等于卷积核数量！

对多通道图像做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。

two_stage: 5. Faster R-CNN

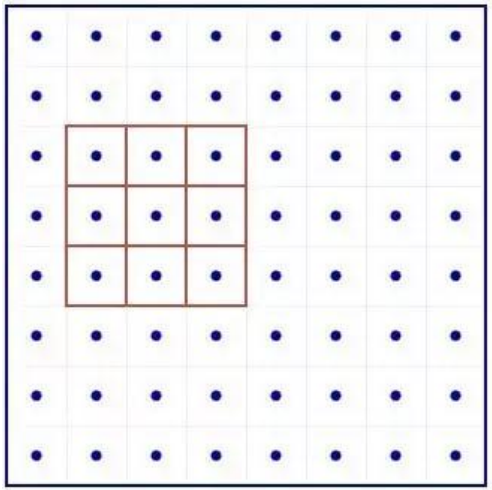
RPN

区域生成网络（RPN）将前面卷积网络的输出特征图作为输入，比如VGG16的conv5特征图。它在特征图上滑动一个3×3 的卷积核，以使用卷积网络构建与类别无关的候选区域。使用VGG网络提取特征的话，每个 3x3 区域会得到一个512维的特征向量，然后送到两个独立的全连接层，以预测边界框和两个目标分数（是目标或者不是目标）。我们其实可以使用回归器计算单个 目标分数，但为简洁起见，Faster R-CNN 使用只有两个类别的分类器：即带有目标的类别和不带有目标的类别。

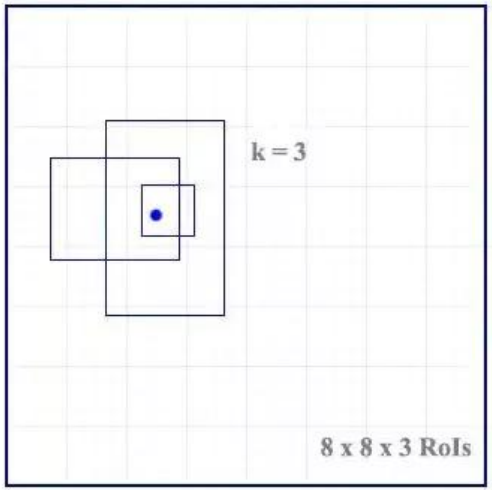


two_stage: 5. Faster R-CNN

对于特征图中的每一个位置，RPN 会做 k 次预测。因此，RPN 将输出 4×k 个坐标和每个位置上 2×k 个得分。下图展示了 8×8 的特征图，且有一个 3×3 的卷积核执行运算，它最后输出 8×8×3 个 ROI（其中 k=3）。下图（右）展示了单个位置的 3 个候选区域。



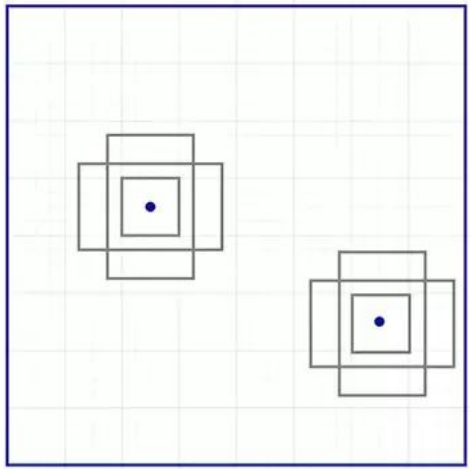
8 x 8 feature maps



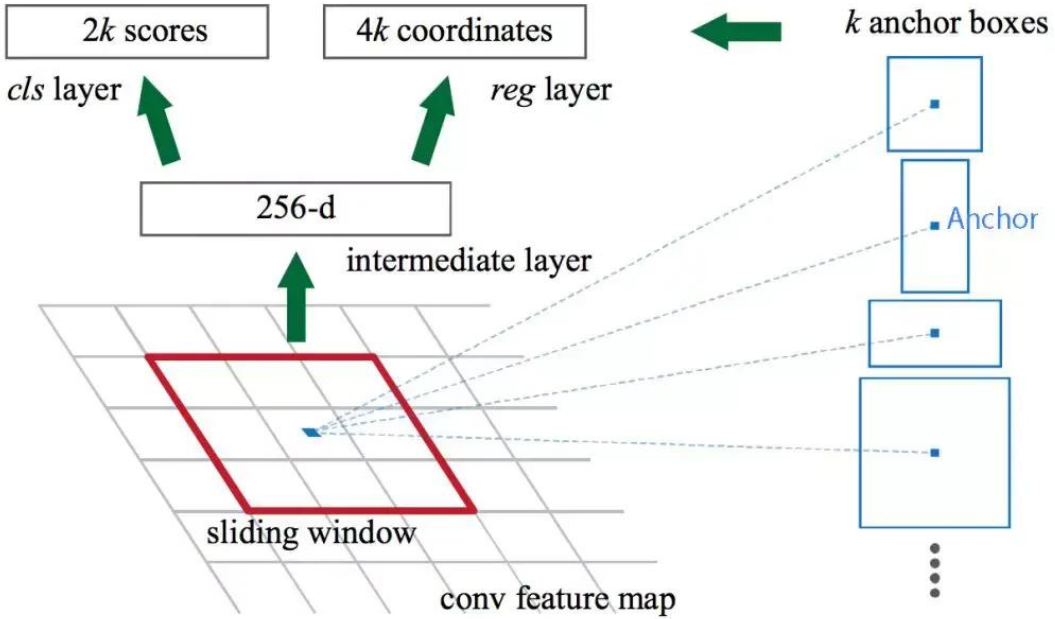
3 proposals for each location

上图每个位置有 3 种猜想，由于只需要一个正确猜想，因此最初的猜想最好涵盖不同的形状和大小。Faster R-CNN 不会创建随机边界框。相反，它会预测一些与左上角名为锚点的参考框相关的偏移量（如δ x、δ y）。要对每个位置进行 k 个预测，我们需要以每个位置为中心的 k 个锚点。这些锚点是精心挑选的，因此它们是多样的，且覆盖具有不同比例和宽高比的现实目标。这使得我们可以以更好的猜想来指导初始训练，并允许每个预测专门用于特定的形状。每个预测与特定锚点相关联，但不同位置共享相同形状的锚点。

two_stage: 5. Faster R-CNN



Faster R-CNN 使用更多的锚点。它部署 9 个锚点框：3 个不同宽高比的 3 个不同大小的锚点框。每一个位置使用 9 个锚点，每个位置会生成 2×9 个目标分数和 4×9 个坐标。



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。

全部anchors（800/16*600/16*9）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

two_stage: 5. Faster R-CNN

FAST RCNN:

```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
for ROI in ROIs
    patch = roi_pooling(feature_maps, ROI)
    results = detector2(patch)
```

FASTER RCNN:

```
feature_maps = process(image)
ROIs = rpn(feature_maps)
for ROI in ROIs
    patch = roi_pooling(feature_maps, ROI)
    class_scores, box = detector(patch)
    class_probabilities = softmax(class_scores)
```

two_stage: 6. R-FCN

之前的Faster RCNN对Fast RCNN产生region porposal的问题给出了解决方案，并且在RPN和Fast RCNN网络中实现了卷积层共享。

但是这种共享仅仅停留在第一卷积部分，RoIpooling及之后的部分没有实现完全共享，可以当做是一种“部分共享”，这导致两个损失：1.信息损失，精度下降。2.由于后续网络部分不共享，导致重复计算全连接层等参数，时间代价过高。(另外还需要多说一句，全连接层计算量是要大于全卷积层的)

因此RFCN（Region-based fully convolutional network）试图以Faster RCNN和FCN为基础进行改进。

FASTER RCNN:

```
feature_maps = process(image)
ROIs = rpn(feature_maps)
for ROI in ROIs
    patch = roi_pooling(feature_maps, ROI)
    class_scores, box = detector(patch)
    class_probabilities = softmax(class_scores)
```

R-FCN:

```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
score_maps = compute_score_map(feature_maps)
for ROI in ROIs
    V = region_roi_pool(score_maps, ROI)
    class_scores, box = average(V)                # Much simpler!
    class_probabilities = softmax(class_scores)
```

R-FCN 通过减少每个 ROI 所需的工作量实现加速。上面基于区域的特征图与 ROI 是独立的，可以在每个 ROI 之外单独计算。剩下的工作就比较简单了，因此 R-FCN 的速度比 Faster R-CNN 快。

two_stage: 6. R-FCN

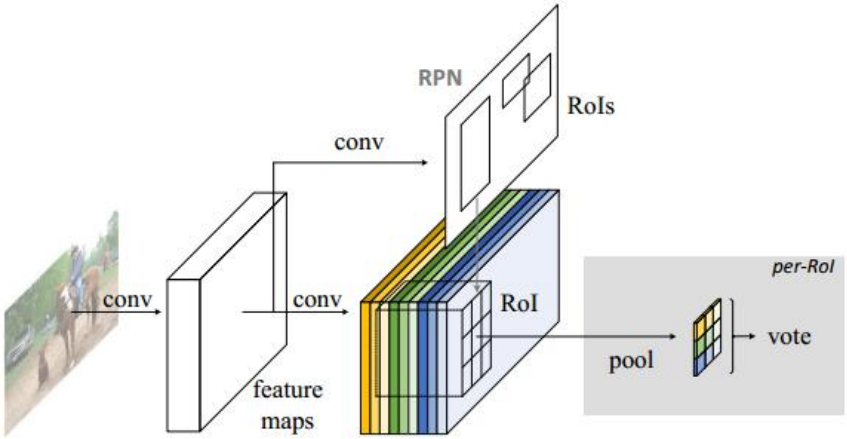
第一个问题，如何改进不完全共享问题

FCN（Fully convolutional network）针对不完全共享问题进行了改进，即：将一般的backbone网络中用于分类的全连接层替换为全卷积层，这样一来整个网络结构均是由卷积层构成，因而称为全卷积网络。

第二个问题，目标检测的需求

很显然，目标检测问题包括两个子问题：第一是确定物体种类，第二是确定物体位置，确定物体种类时我们希望保持位置不敏感性（translation invariance也就是说不管物体出现在哪个位置都能正确分类）以及保持位置敏感性（translation variance我们当然希望不论物体发生怎样的位置变化都能确定物体位置）

这两个需求看起来比较矛盾，RFCN做出了一个折中，实际上也不算折中吧，就是这样一个问题：我们知道全卷积网络提取特征非常强，因此用于物体分类很nice，但是普通的卷积网络只关注特征，并不关注位置信息，不能直接用于检测。所以RFCN在FCN网络中引入了一个概念“position sensitive score map”位置敏感得分图，用来保证全卷积网络对物体位置的敏感性



two_stage: 6. R-FCN

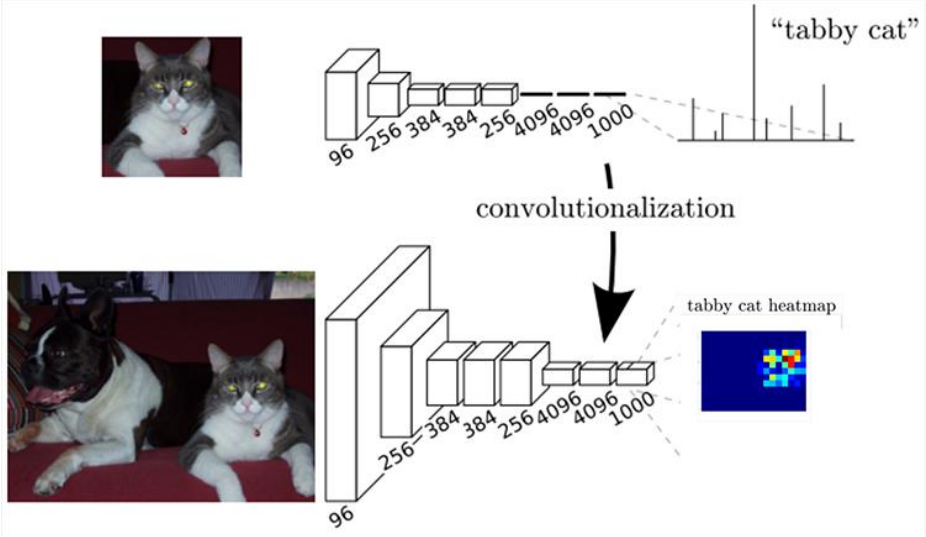
CNN与FCN

通常cnn网络在卷积之后会接上若干个全连接层，将卷积层产生的特征图（feature map）映射成为一个固定长度的特征向量。一般的CNN结构适用于图像级别的分类和回归任务，因为它们最后都期望得到输入图像的分类的概率，如ALexNet网络最后输出一个1000维的向量表示输入图像属于每一类的概率。

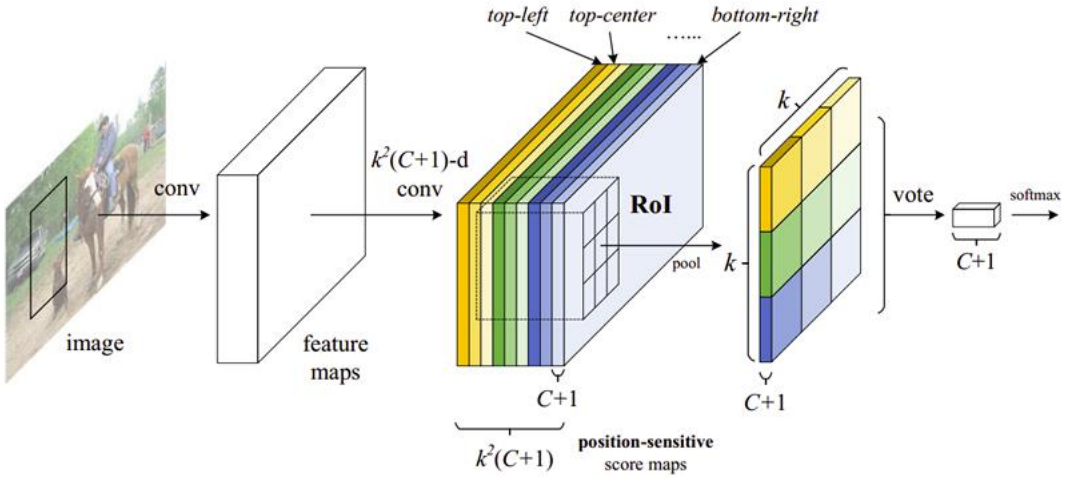
FCN对图像进行像素级的分类，从而解决了语义级别的图像分割问题。与经典的CNN在卷积层使用全连接层得到固定长度的特征向量进行分类不同，FCN可以接受任意尺寸的输入图像，采用反卷积层对最后一个卷基层的特征图（feature map）进行上采样，使它恢复到输入图像相同的尺寸，从而可以对每一个像素都产生一个预测，同时保留了原始输入图像中的空间信息，最后在上采样的特征图进行像素的分类。

全卷积网络(FCN)是从抽象的特征中恢复出每个像素所属的类别。即从图像级别的分类进一步延伸到像素级别的分类。

FCN将传统CNN中的全连接层转化成一个个的卷积层。如下图所示，在传统的CNN结构中，前5层是卷积层，第6层和第7层分别是一个长度为4096的一维向量，第8层是长度为1000的一维向量，分别对应1000个类别的概率。FCN将这3层表示为卷积层，卷积核的大小(通道数，宽，高)分别为（4096,7,7）、（4096,1,1）、（1000,1,1）。所有的层都是卷积层，故称为全卷积网络。



two_stage: 6. R-FCN



有两个关键层：

1）包含多个 Score Map 的卷积层；

把目标分割成了 k*k 个部分（比如3*3），每个部分映射到一张 Score Map 上，每个 Score Map 对应目标的一部分（如上图中的 top-left 左上角的 1/9）。

最终得到 k*k 个Score Map，每一个 Map通道数为 分类个数 C+1。

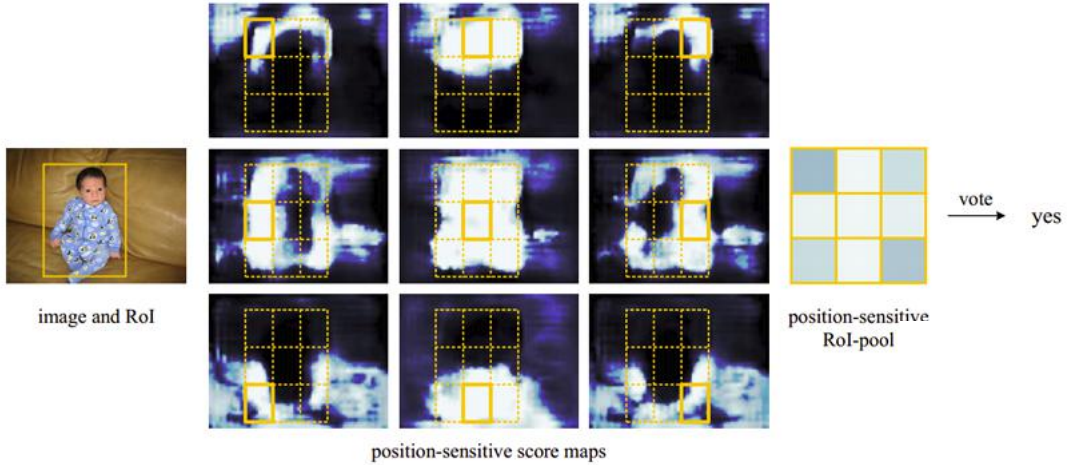
2）一个 ROI Pooling 层；

这个 ROI 层仅针对上面的其中一个 Score Map 执行 Pooling 操作，重新排列成 k*k，通道数为 C+1。

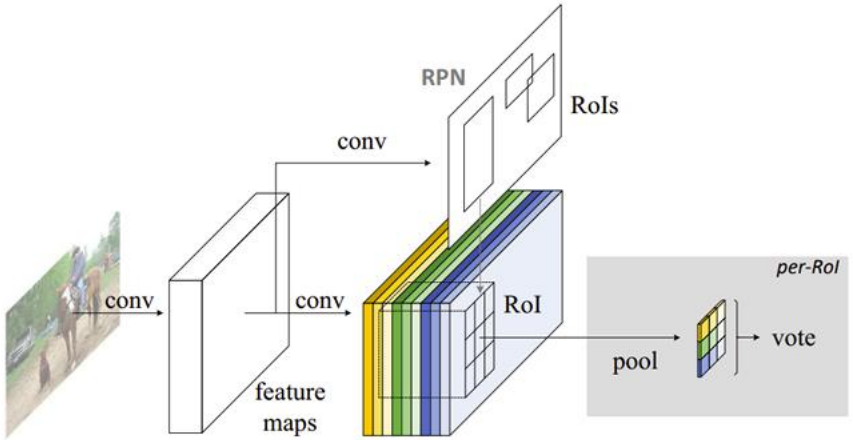
ROI Pooling 层通过 k*k 个 Part 进行投票，得到分类结果。

two_stage: 6. R-FCN

Score Map 和 ROI Pooling 层的工作方式示意如下：



R-FCN 沿用了 Faster RCNN 的网络结构，通过 RPN 生成 Proposal，RPN 层与 Detection 共享前面的特征层：



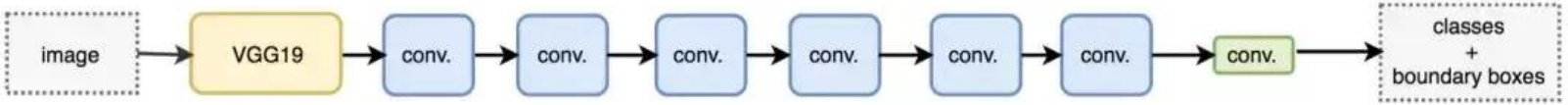
one_stage

Faster R-CNN 中有一个专用的候选区域网络RPN。基于区域的检测器是很准确的，但需要付出代价。Faster R-CNN 在 PASCAL VOC 2007 测试集上每秒处理 7 帧的图像（7 FPS）。作为替代，我们是否需要一个分离的候选区域步骤？我们可以直接在一个步骤内得到边界框和类别吗？

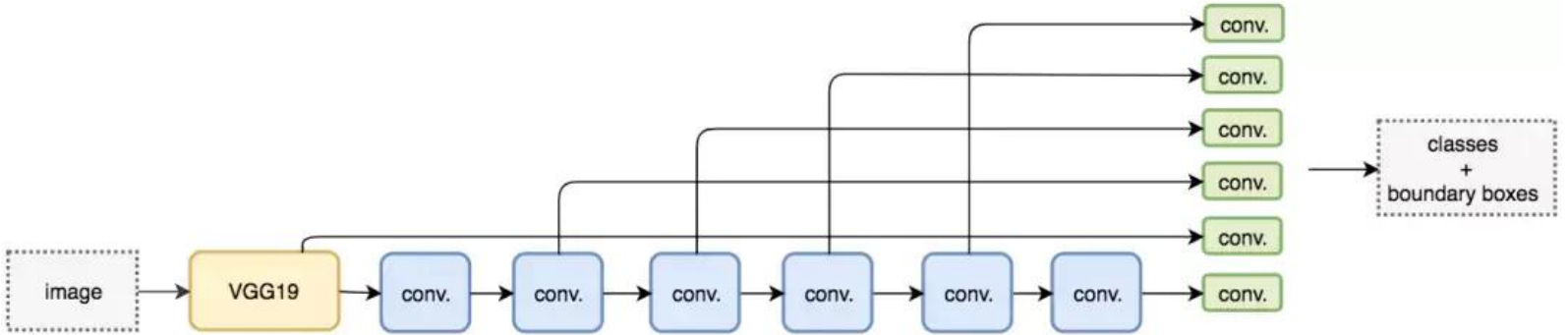
让我们再看一下滑动窗口检测器。我们可以通过在特征图上滑动窗口来检测目标。对于不同的目标类型，我们使用不同的窗口类型。以前的滑动窗口方法的致命错误在于使用窗口作为最终的边界框，这就需要非常多的形状来覆盖大部分目标。更有效的方法是将窗口当做初始猜想，这样我们就得到了从当前滑动窗口同时预测类别和边界框的检测器。以下是 4 个锚点（绿色）和 4 个对应预测（蓝色），每个预测对应一个特定锚点.

one_stage: SSD

SSD（Single-Shot MultiBox Detector），使用 VGG19 网络作为特征提取器（和 Faster R-CNN 中使用的 CNN 一样）的单次检测器。我们在该网络之后添加自定义卷积层（蓝色），并使用卷积核（绿色）执行预测。

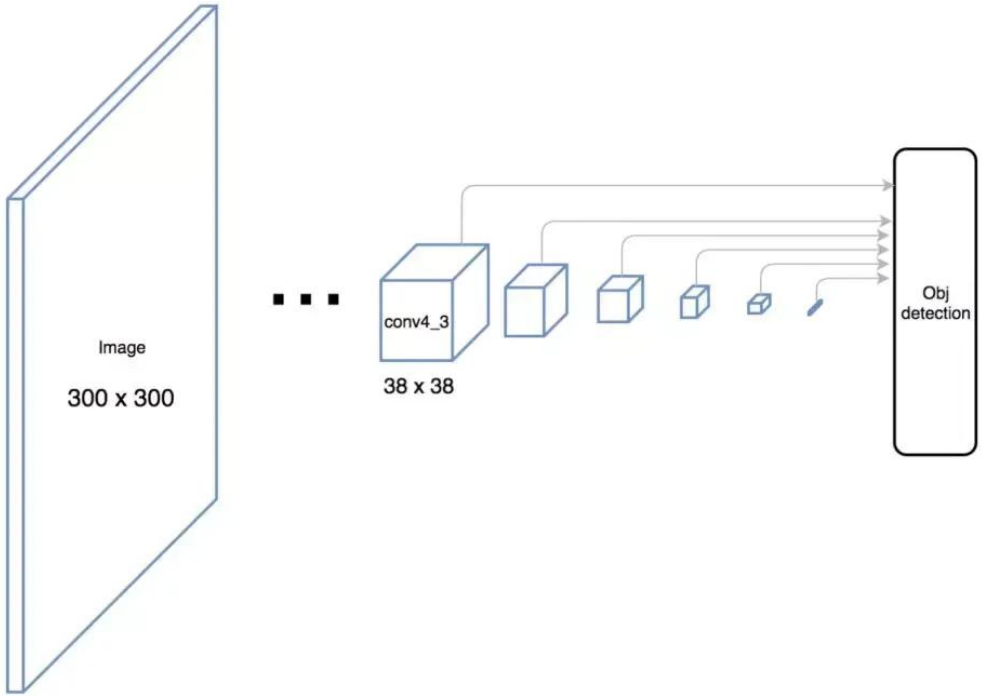


然而，卷积层降低了空间维度和分辨率。因此上述模型仅可以检测较大的目标。为了解决该问题，我们从多个特征图上执行独立的目标检测。



one_stage: SSD

SSD 使用卷积网络中较深的层来检测目标。如果我们按接近真实的比例重绘上图，我们会发现图像的空间分辨率已经被显著降低，且可能已无法定位在低分辨率中难以检测的小目标。如果出现了这样的问题，我们需要增加输入图像的分辨率。



one_stage: DSSD

DSSD：DeconvolutionalSingleShotDetector

【DSSD创新点】

1、使用ResNet101作为特征提取网络，提取更深层次的特征。

2、提出基于top down的网络结构，并用反卷积代替传统的双线性插值上采样。

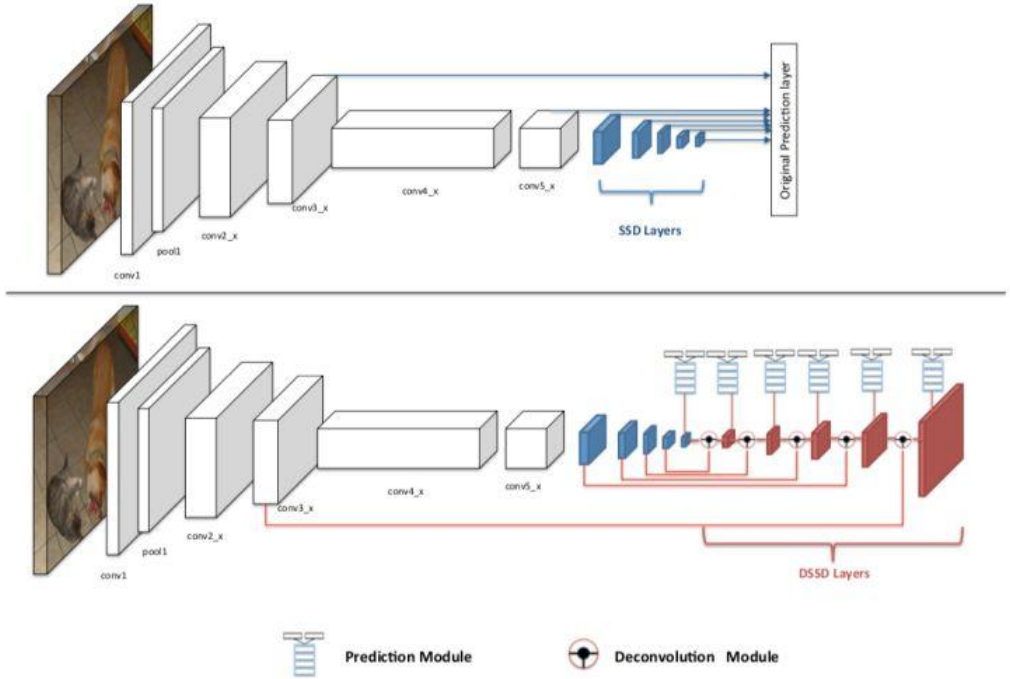
3、在预测阶段引入残差单元，优化候选框回归和分类任务输入的特征图

DSSD和SSD的网络结构对比如下，可以看出DSSD在SSD的基础上做了3处改进：

1）ResNet101作为backbone；

2）新增Deconvoluational Module；

3）新增Prediction Module

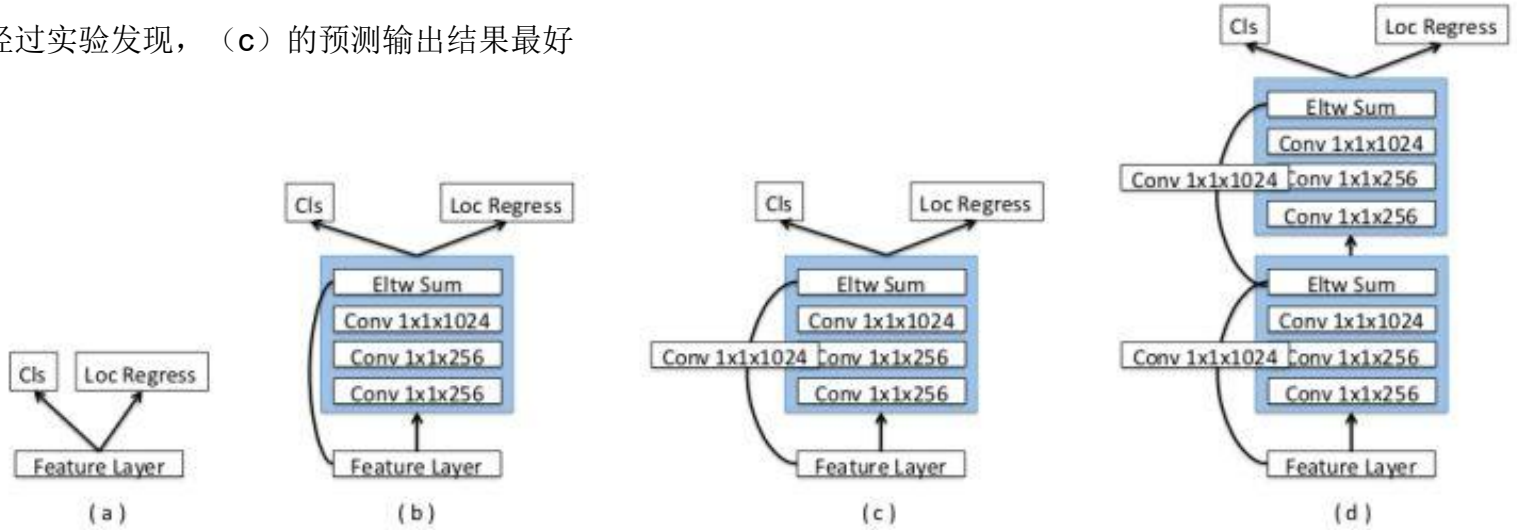


one_stage: DSSD

Prediction Module的结构如下图所示：

- （a）为SSD所使用的预测方法，直接在多特征图上使用两个 3×3 的卷积分别做分类和边框回归。
- （b）是在（a）的基础上增加了一个残差单元。
- （c）为作者改进的只含一个残差单元的预测模型，在残差旁路将原来的特征图用的卷积核做处理后与网络主干道的特征图做通道间加法。
- （d）包含两个残差单元的预测模型。

作者经过实验发现，（c）的预测输出结果最好



one_stage: DSSD

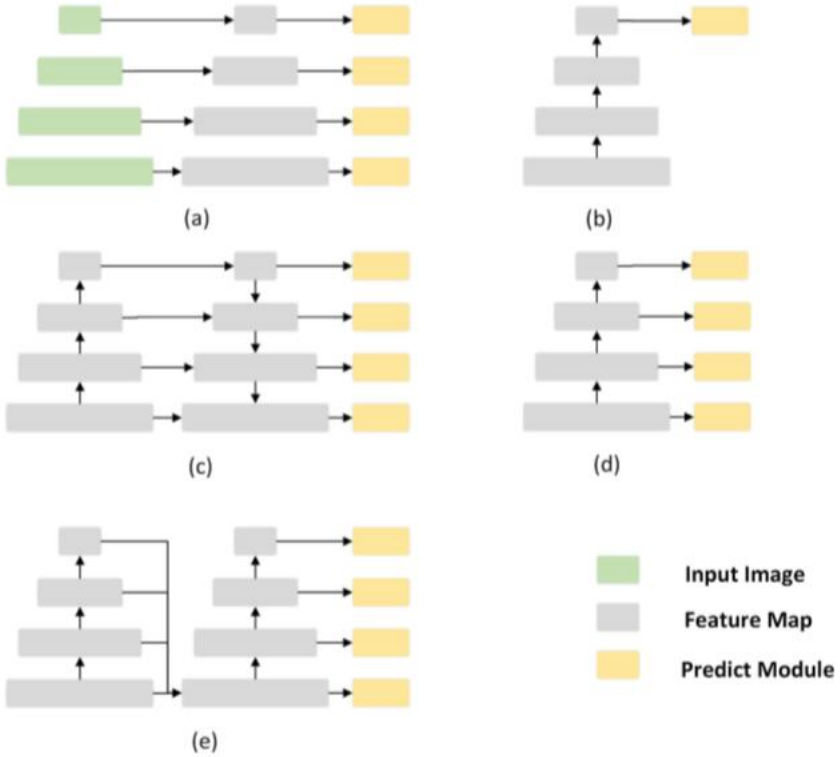
Method	network	mAP	With BN layers		BN layers removed		# Proposals	GPU	Input resolution
			FPS	batch size	FPS	batch size			
Faster R-CNN [24]	VGG16	73.2	7	1	-	-	6000	Titan X	$\sim 1000 \times 600$
Faster R-CNN [14]	Residual-101	76.4	2.4	1	-	-	300	K40	$\sim 1000 \times 600$
R-FCN [3]	Residual-101	80.5	9	1	-	-	300	Titan X	$\sim 1000 \times 600$
SSD300*[18]	VGG16	77.5	46	1	-	-	8732	Titan X	300×300
SSD512*[18]	VGG16	79.5	19	1	-	-	24564	Titan X	512×512
SSD321	Residual-101	77.1	11.2	1	16.4	1	17080	Titan X	321×321
SSD321	Residual-101	77.1	18.9	15	22.1	44	17080	Titan X	321×321
DSSD321	Residual-101	78.6	9.5	1	11.8	1	17080	Titan X	321×321
DSSD321	Residual-101	78.6	13.6	12	15.3	36	17080	Titan X	321×321
SSD513	Residual-101	80.6	6.8	1	8.0	1	43688	Titan X	513×513
SSD513	Residual-101	80.6	8.7	5	11.0	16	43688	Titan X	513×513
DSSD513	Residual-101	81.5	5.5	1	6.4	1	43688	Titan X	513×513
DSSD513	Residual-101	81.5	6.6	4	6.3	12	43688	Titan X	513×513

Table 7: **Comparison of Speed & Accuracy on PASCAL VOC2007 `test`.**

one_stage: FSSD

FSSD：Feature Fusion Single Shot Multibox Detector

主要贡献：提出一个特征融合模块（Feature Fusion Module）。
其中：方式c是FPN的方式，方式d是SSD中采用的方式，e是本文采用的融合方式，就是把网络中某些feature调整为同一szie再 contact，得到一个像素层，以此层为base layer来生成pyramid feature map，作者称之为Feature Fusion Module。该方式与FPN相比，只需要融合一次，较为简单，在融合时方式e采用的时concat，标准的fpn采用的时sum



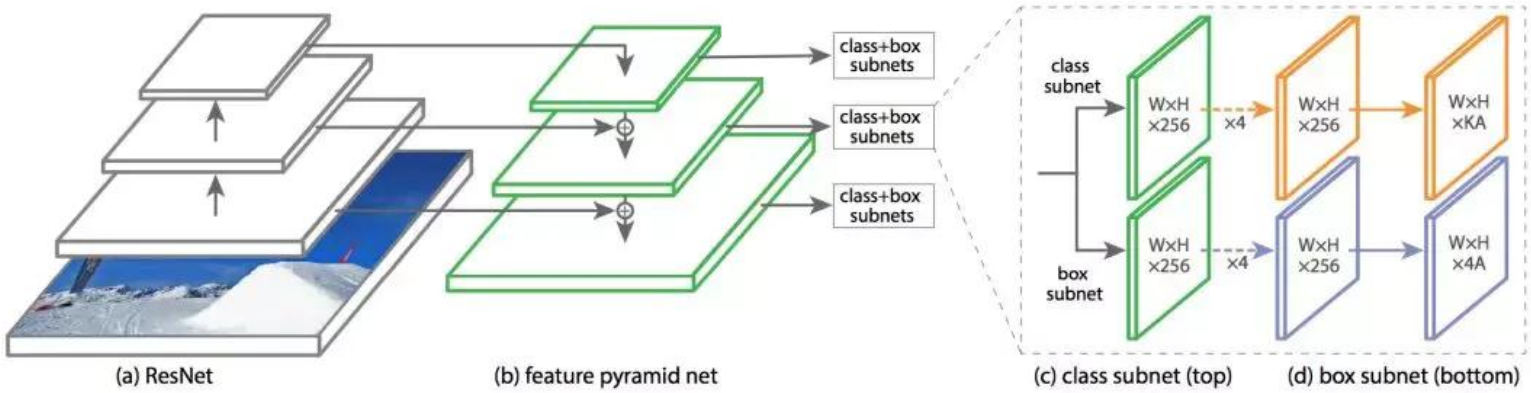
one_stage: FSSD

FSSD：Feature Fusion Single Shot Multibox Detector

Method	data	pre-train	backbone network	speed (*fps*)	GPU	#proposals	input size	mAP
Faster RCNN [28]	07+12	✓	VGGNet	7	Titan X	6000	~ 600 × 1000	73.2
Faster RCNN [28]	07+12	✓	ResNet-101	2.4	K40	300	~ 600 × 1000	76.4
R-FCN [3]	07+12	✓	ResNet-50	-	-	300	~ 600 × 1000	77.0
R-FCN [3]	07+12	✓	ResNet-101	5.8	K40	300	~ 600 × 1000	79.5
YOLOv2 [27]	07+12	✓	Darknet-19	81	Titan X	-	352 × 352	73.7
SSD300S† [31]	07+12	✗	VGGNet	46	Titan X	8732	300 × 300	69.6
SSD300* [21]	07+12+COCO	✓	VGGNet	46	Titan X	8732	300 × 300	81.2
SSD300 [21]	07+12	✓	VGGNet	46	Titan X	8732	300 × 300	77.2
SSD300 [21]	07+12	✓	VGGNet	85	1080Ti	8732	300 × 300	77.2
SSD512 [21]	07+12	✓	VGGNet	19	Titan X	24564	512 × 512	78.5
SSD512* [21]	07+12+COCO	✓	VGGNet	19	Titan X	24564	512 × 512	83.2
DSOD300 [31]	07+12	✗	DS/64-192-48-1	17.4	Titan X	-	300 × 300	77.7
DSOD300* [31]	07+12+COCO	✗	DS/64-192-48-1	17.4	Titan X	-	300 × 300	81.7
DSSD321 [7]	07+12	✓	ResNet-101	9.5	Titan X	17080	321 × 321	78.6
DSSD513 [7]	07+12	✓	ResNet-101	5.5	Titan X	43688	321 × 321	81.5
RSSD300 [16]	07+12	✓	VGGNet	35	Titan X	8732	300 × 300	78.5
RSSD512 [16]	07+12	✓	VGGNet	16.6	Titan X	24564	512 × 512	80.8
FSSD300S†	07+12	✗	VGGNet	65.8	1080Ti	8732	300 × 300	72.7
FSSD300	07+12	✓	VGGNet	65.8	1080Ti	8732	300 × 300	78.8
FSSD300*	07+12+COCO	✓	VGGNet	65.8	1080Ti	8732	300 × 300	**82.7**
FSSD512	07+12	✓	VGGNet	35.7	1080Ti	24564	512 × 512	80.9
FSSD512*	07+12+COCO	✓	VGGNet	35.7	1080Ti	24564	512 × 512	**84.5**

one_stage: RetianNet

RetianNet是基于 ResNet、FPN以及利用 Focal loss 构建的



Focal loss主要是为了解决one-stage目标检测中正负样本比例严重失衡的问题。该损失函数降低了大量简单负样本在训练中所占的权重，也可理解为一种困难样本挖掘。

类别不平衡会损害性能。SSD 在训练期间重新采样目标类和背景类的比率，这样它就不会被图像背景淹没。Focal loss（FL）采用另一种方法来减少训练良好的类的损失。因此，只要该模型能够很好地检测背景，就可以减少其损失并重新增强对目标类的训练。我们从交叉熵损失 （Cross Entroy Loss）开始，并添加一个权重来降低高可信度类的交叉熵。

one_stage: RetianNet

在一张输入image中，目标占的比例一般都远小于背景占的比例，所以两类example中以negative为主，这引发了两个问题：

negative example过多造成它的loss太大，以至于把positive的loss都淹没掉了，不利于目标的收敛；
大多negative example不在前景和背景的过渡区域上，分类很明确(这种易分类的negative称为easy negative)，训练时对应的背景类score会很大，换个角度看就是单个example的loss很小，反向计算时梯度小。梯度小造成easy negative example对参数的收敛作用很有限，我们更需要loss大的对参数收敛影响也更大的example，即hard positive/negative example。

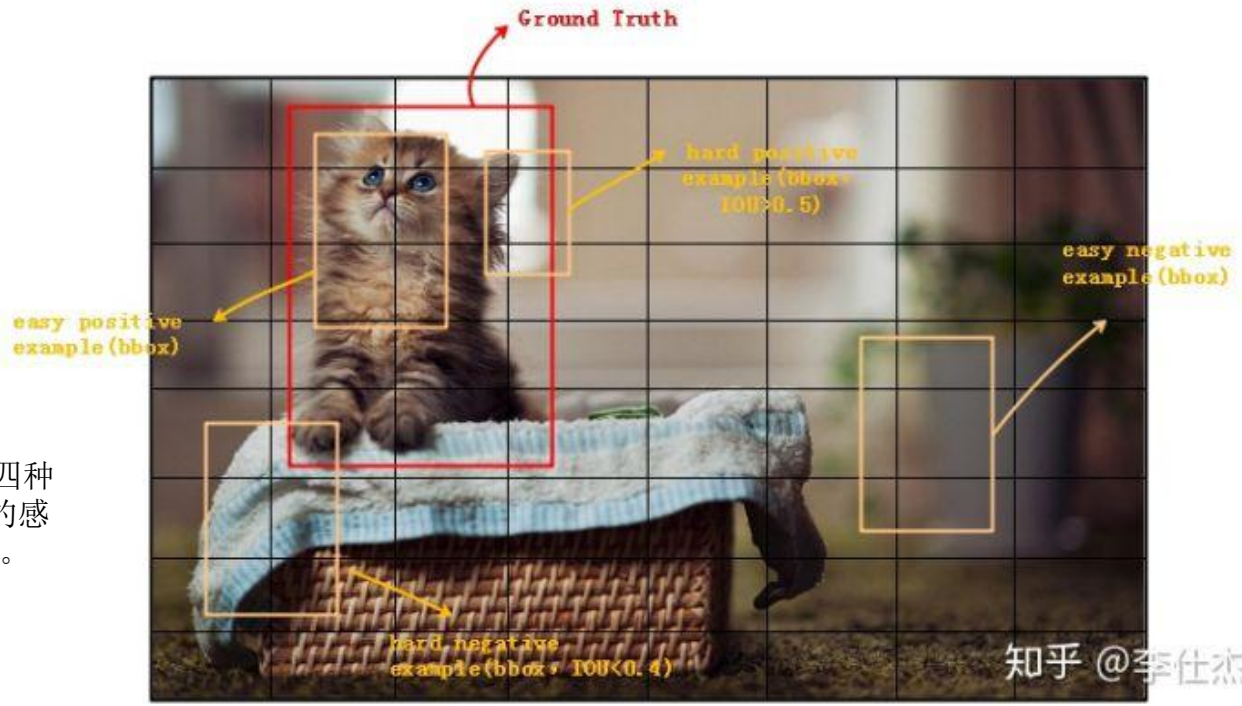
这里要注意的是前一点我们说了negative的loss很大，是因为negative的绝对数量多，所以总loss大；后一点说easy negative的loss小，是针对单个example而言。

Faster RCNN的两级结构可以很好的规避上述两个问题：

会根据前景score的高低过滤出最有可能是前景的example (1K~2K个)，因为依据的是前景概率的高低，就能把大量背景概率高的easy negative给过滤掉，这就解决了前面的第2个问题；
会根据IOU的大小来调整positive和negative example的比例，比如设置成1：3，这样防止了negative过多的情况(同时防止了easy negative和hard negative)，就解决了前面的第1个问题。
所以Faster RCNN的准确率高。

one_stage: RetianNet

OHEM是近年兴起的另一种筛选example的方法，它通过对loss排序，选出loss最大的example来进行训练，这样就能保证训练的区域都是hard example。这个方法有个缺陷，它把所有的easy example都去除掉了，造成easy positive example无法进一步提升训练的精度



hard positvie、hard negative、easy positive、easy negative四种example的示意图，可以直观的感受到easy negativa占了大多数。

one_stage: RetianNet

Focal loss是在交叉熵损失函数基础上进行的修改，首先回顾二分类交叉上损失：

$$L = -ylog(y') - (1-y)log(1-y')$$

y' 是经过激活函数的输出，所以在0-1之间。可见普通的交叉熵对于正样本而言，输出概率越大损失越小。对于负样本而言，输出概率越小则损失越小。此时的损失函数在大量简单样本的迭代过程中比较缓慢且可能无法优化至最优。

$$FL(p_t) = \alpha_t(1-p_t)^{\gamma}log(p_t)$$

Focal Loss通过调整loss的计算公式使单级结构达到和Faster RCNN一样的准确度。

p_t 是不同类别的分类概率， γ 是个大于0的值， α_t 是个[0，1]间的小数， γ 和 α_t 都是固定值，不参与训练。从表达式可以看出：

1. 无论是前景类还是背景类， p_t 越大，权重 $(1-p_t)$ 就越小。也就是说easy example可以通过权重进行抑制。换言之，当某样本类别比较明确些，它对整体loss的贡献就比较少；而若某样本类别不易区分，则对整体loss的贡献就相对偏大。这样得到的loss最终将集中精力去诱导模型去努力分辨那些难分的目标类别，于是就有效提升了整体的目标检测准度。
2. α_t 用于调节positive和negative的比例，前景类别使用 α_t 时，对应的背景类别使用 $1-\alpha_t$ ；
3. γ 和 α_t 的最优值是相互影响的，所以在评估准确度时需要把两者组合起来调节。作者在论文中给出 $\gamma = 2$ 、 $\alpha_t = 0.25$ 时，ResNet-101+FPN作为backbone的结构有最优的性能。

α系数，它能够使得focal loss对不同类别更加平衡。

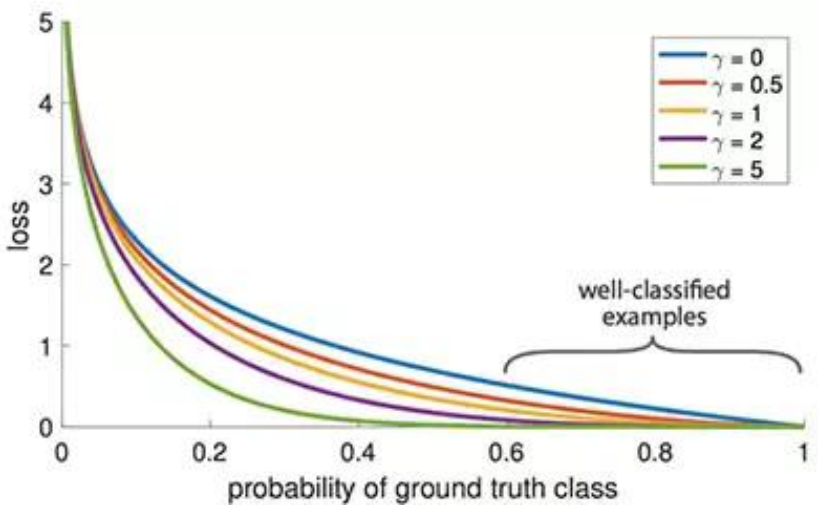
one_stage: RetianNet

$$CE(p_t) = -\log(p_t)$$
$$FL(p_t) = -(1-p_t)^{\gamma}\log(p_t)$$

where

p_t is the predicted class probability for ground truth.
$\gamma > 0$

例如，令 γ = 0.5, 经良好分类的样本的 Focal Loss 趋近于 0。



1. 训练时FPN每一级的所有example都被用于计算Focal Loss，loss值加到一起用来训练；
2. 测试时FPN每一级只选取score最大的1000个example来做nms；
3. 整个结构不同层的head部分(图 2 的c和d部分)共享参数，但分类和回归分支间的参数不共享
4. 分类分支的最后一级卷积的bias初始化成前面提到的 $-log((1-\pi)/\pi)$；