

第二次组会汇报：SPPNet 部分内容+Fast R-CNN 论文讲解

汇报人：王振宽

汇报时间：2022/11/18

一、前言回顾：

1.R-CNN 的步骤

- 找出图片中可能存在目标的候选区域 region proposal
- 进行图片大小调整为了适应 AlexNet 网络的输入图像的大小 227×227 ，通过 CNN 对候选区域提取特征向量，2000 个建议框的 CNN 特征组合成 2000×4096 维矩阵
- 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
- 分别对 2000×20 维矩阵中每一列即每一类进行非极大值抑制 (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
- 修正 bbox，对 bbox 做回归微调

但是很可惜，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 仍会有严重的瓶颈：

- 速度瓶颈：重复为每个 region proposal 提取特征是极其费时的，Selective Search 对于每幅图片产生 2K 左右个 region proposal，也就是意味着一幅图片需要经过 2K 次的完整的 CNN 计算得到最终的结果。
- 性能瓶颈：对于所有的 region proposal 防缩到固定的尺寸会导致我们不期望看到的几何形变，而且由于速度瓶颈的存在，不可能采用多尺度或者是大量的数据增强去训练模型。

2. SPP Net 的步骤

为了解决 R-CNN 中速度慢问题，何凯明大神提出 SPPNet。

- SPP Net 的第一个贡献就是在最后一个卷积层后，接入了金字塔池化层，保证传到下一层全连接层的输入固定。

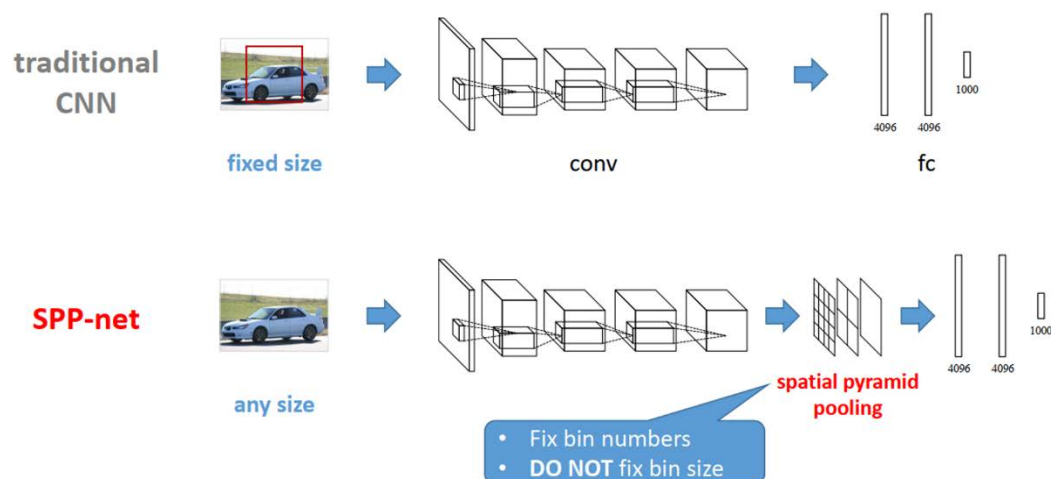


图 1：SPPNet 与传统 CNN 的对比

在普通的 CNN 机构中，输入图像的尺寸往往是固定的（比如 224*224 像素），输出则是一个固定维数的向量。SPP Net 在普通的 CNN 结构中加入了 ROI 池化层（ROI Pooling），使得网络的输入图像可以是任意尺寸的，输出则不变，同样是一个固定维数的向量。

- SPP Net 的第二个贡献就是减少卷积运算

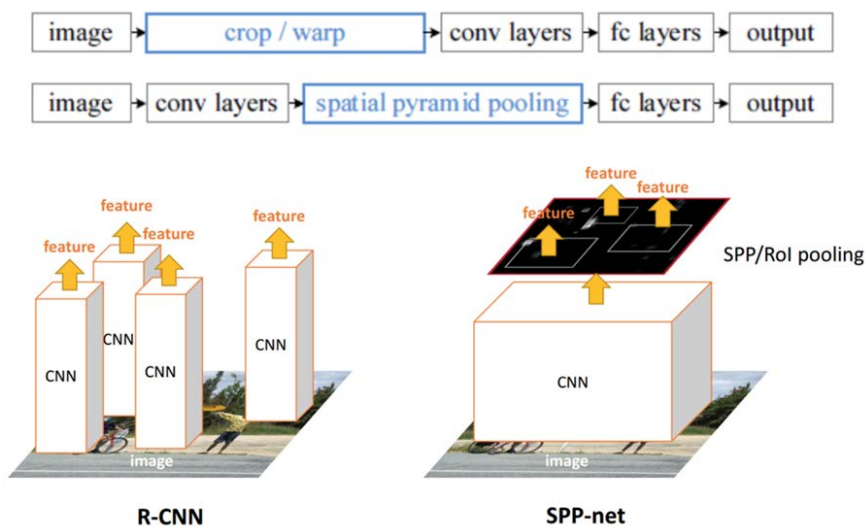


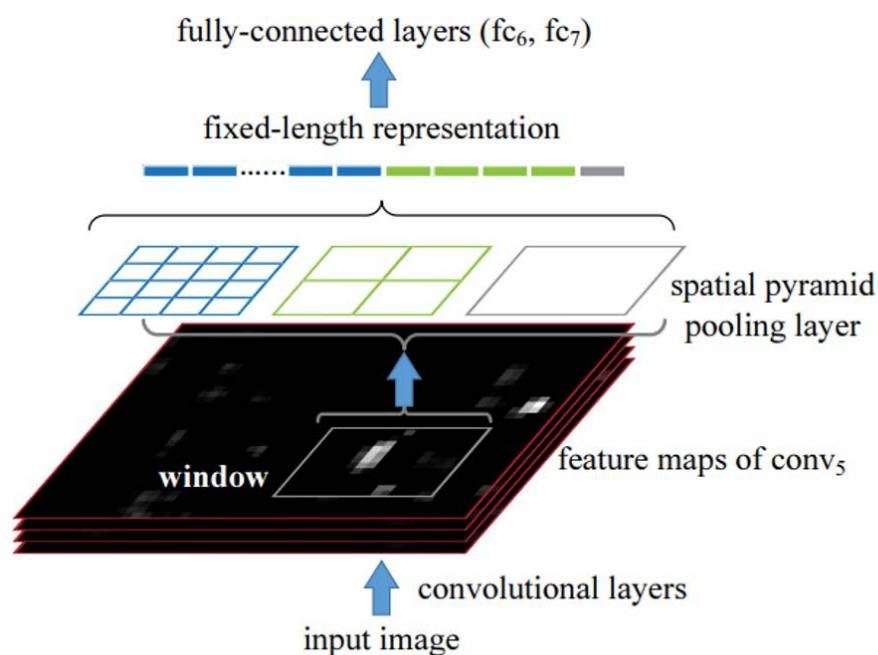
图 2：与 R-CNN 的对比

对于多个候选区域依次进入特征提取网络进行图像特征提取，SPPNet 采用直接将全图输入到卷积神经网络中提取图像特征，然后在图像特征图上找到候选区域的图像特征。此外，采用 Spatial Pyramid Pooling 将不同尺寸的候选区域特征转化为固定尺寸的值，避免了 RCNN 采用的缩放操作。

R-CNN 要对每个区域计算卷积，而 SPPNet 只需要计算一次卷积，从而节省了大量的计算时间，比 R-CNN 有一百倍左右的提速。

SPPNet 做目标检测的主要步骤：

- 利用 SS 算法从原图中生成 2000 个左右的候选窗口
- 卷积操作：将输入图像进行卷积操作，得到整个图像的卷积特征
- 特征提取：对原始图像的各种候选框，需要在卷积特征中找到相应的位置框，SPPNet 不再做区域大小的归一化，而是直接使用 ROI 池化层对位置框中的卷积提取特征
- 分类与回归：类似 R-CNN，利用 SVM 基于上面的特征训练分类器模型，用边框回归来微调候选框的位置。



SPP-Net 中的难点:

虽然总体流程还是 Selective Search 得到候选区域->CNN 提取 ROI 特征->类别判断->位置精修, 但是由于所有 ROI 的特征直接在 feature map 上提取, 大大减少了卷积操作, 提高了效率。但是依旧有一个难点:

- 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)

作者发现,卷积后对应的位置并不会发生改变, 每个卷积层会匹配响应的区域。如下图所示:

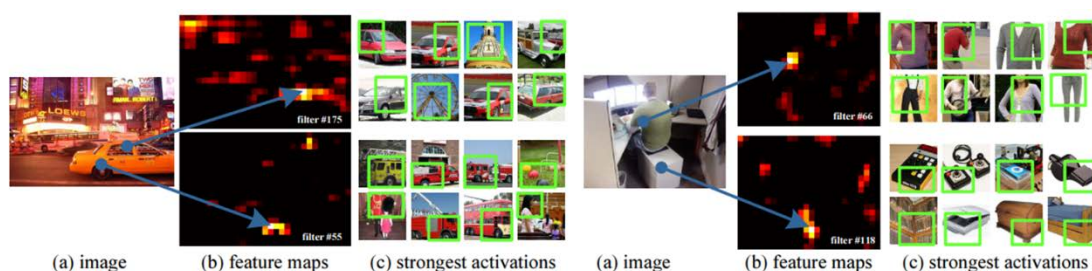


Figure 2: Visualization of the feature maps. (a) Two images in Pascal VOC 2007. (b) The feature maps of some conv₅ filters. The arrows indicate the strongest responses and their corresponding positions in the images. (c) The ImageNet images that have the strongest responses of the corresponding filters. The green rectangles mark the receptive fields of the strongest responses.

https://tong.cslu.net/webinar_45036944

接下来讲解如何映射:

这部分的计算其实就是感受野大小的计算。卷积神经网络 CNN 中, 某一层的输

出结果中一个元素所对应的输入层的区域大小，被称作感受野，感受野的大小是由 kernel size、stride、padding、output size 一起决定的。

卷积过程中的词语含义：

28×28 的输入图片，输入图片尺寸为 $W = 28$ ；

5×5 的卷积核尺寸，卷积核大小 $K = 5$ ；

滑动步长为 stride。

当 $W = 28$ ， $K = 5$ ，stride=1 时，可以得到 24×24 的输出。

我们可以推出公式：边长 = $(W - K) / S + 1$

注意，有时候为了控制输出的隐藏层空间分布会在输入层外围做零填充，假设填充 P 个像素，此时：边长 = $(W - K + 2P) / S + 1$

其中， W 是输入特征的大小， K 是卷积核的大小， P 是填充大小， S 是步长。

那么我们该如何计算卷积层的输入呢？即我们如何计算前一层的感受野的大小？

令输出边长为 Y ，我们得到 W 的表达式为：

$$W = (Y - 1) \times S - 2P + K$$

这里我们详细介绍一下坐标的推导：

记某卷积层的 kernel size 为 k ，其输入大小为 $w \times h$ ，则输出大小为：

$$w' = (w - k + 2p) / s + 1$$

$$h' = (h - k + 2p) / s + 1$$

那么输出 feature map 上一点坐标为 (i', j') 时，其在输入平面上对应一个大小为

k 的窗口，记这个窗口的左上角坐标为 (i, j) ，那么有：

$$i + p = i' s \Rightarrow i = i' s - p$$

$$j + p = j' s \Rightarrow j = j' s - p$$

坐标为整数。可以验证一下，输出 feature map 上最右端点横坐标为 $w' - 1$ ，对应

输入平面的最右端窗口，其左上角横坐标，根据 $i + p = i's \Rightarrow i = i's - p$ 式为：

$(w'-1)s - p$ ，窗口大小为 k ，那么右上角的横坐标为 $(w'-1)s - p + k - 1$ ，同时也是输入平面最右端的像素（padding 后），故也等于 $w + p - 1$ ，所以 $w + p - 1 = (w'-1)s - p + k - 1$ ，与 $w' = (w - k + 2p) / s + 1$ 吻合。

于是这个窗口的中心点坐标为：

$$p_x = i + (k - 1) / 2 = i's + (k - 1) / 2 - p$$

$$p_y = j + (k - 1) / 2 = j's + (k - 1) / 2 - p$$

我们参考吴恩达在 Coursera 讲解中对符号的定义：

在第 i 层的坐标： p_i

第 i 层的步长： s_i


第 i 层的卷积核的大小： k_i

第 i 层填充的大小： padding

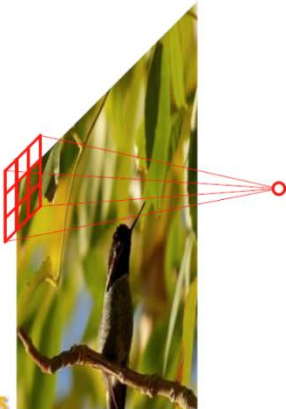
计算公式总结为：

- 对于卷积层或者池化层： $p_i = s_i \times p_{i+1} + [(k_i - 1) / 2 - padding]$

下面看一下何凯明演讲的 PPT：



Receptive Field




How to compute **the center of the receptive field**

- A simple solution
 - For each layer, pad $\lfloor F/2 \rfloor$ pixels for a filter size F (e.g., pad 1 pixel for a filter size of 3)
 - On each feature map, the response at $(0, 0)$ has a receptive field centered at $(0, 0)$ on the image
 - On each feature map, the response at (x, y) has a receptive field centered at (Sx, Sy) on the image (stride S)
- A general solution
$$i_0 = g_L(i_L) = \alpha_L(i_L - 1) + \beta_L$$

$$\alpha_L = \prod_{p=1}^L S_p$$

$$\beta_L = 1 + \sum_{p=1}^L \left(\prod_{q=p+1}^L S_q \right) \left(\frac{F_p - 1}{2} - P_p \right)$$

See [Karel Lenc & Andrea Vedaldi] "R-CNN minus R", BMVC 2015.



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". ECCV 2014.

简单翻译一下 PPT 上的内容：

问题：如何计算感受野区域的中心？

一个简单的解决方案是：对每一层，填充零元素 $P = \frac{F}{2}$ 向下取整，这样的话，在每一个特征图上坐标为 (0,0) 的响应点的感受野是以输入图像坐标为 (0,0) 的点为中心的，坐标为 (x,y) 的点的感受野是以 (Sx,Sy) 为中心的，S 为步长。

最后给出了计算的通解。

PPT 上的公式推导：

SPP-Net 对上面的坐标对应关系作了一定的简化，简化过程如下：

令每一层的 padding 都为：

$$padding = \left\lfloor \frac{k_i}{2} \right\rfloor \rightarrow p_i = s_i \times p_{i+1} + \left(\frac{k_i - 1}{2} - \left\lfloor \frac{k_i}{2} \right\rfloor \right)$$

当 k_i 为奇数时， $\left(\frac{k_i - 1}{2} - \left\lfloor \frac{k_i}{2} \right\rfloor \right) = 0$ ，所以 $p_i = s_i \times p_{i+1}$ ；

当 k_i 为偶数时， $\left(\frac{k_i - 1}{2} - \left\lfloor \frac{k_i}{2} \right\rfloor \right) = -0.5$ ，所以 $p_i = s_i \times p_{i+1} - 0.5$

我们的 p_i 是像素块的坐标，不能取小数，所以基本上可以认为 $p_i = s_i \times p_{i+1}$

公式得到了极大的化简，得出结论：感受野中心点的坐标 p_i 只与前一层 p_{i+1} 相关。

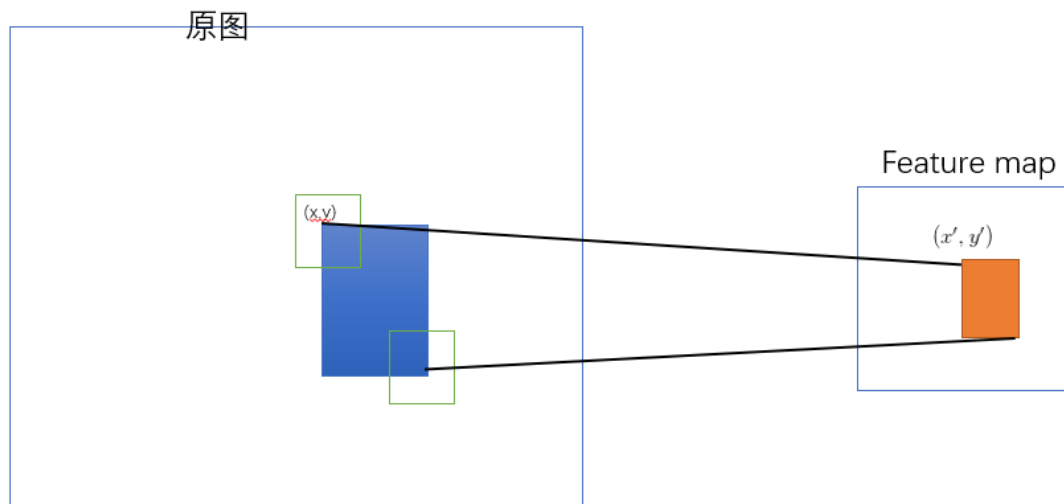
这也是上面的一个简单的计算方法。

PPT 中的下面的通解，其实就是把公式 $p_i = s_i \times p_{i+1} + [(k_i - 1) / 2 - padding]$ 级联消去整合一下而已。

SPP-Net 中 ROI 映射做法详解：

SPP-net 是把原始 ROI 的左上角和右下角 映射到 feature map 上的两个对应点。

有了 feature map 上的两角点就确定了 对应的 feature map 区域(下图中橙色)：



先解释一下两个符号的表示含义：

- $\lceil \cdot \rceil$ 表示向上取整
- $\lfloor \cdot \rfloor$ 表示向下取整

左上角的点 (x, y) 如何映射到 feature map 上的 (x', y') ，使得 (x', y') 在原始图上感受野的中心点与 (x, y) 尽可能接近。

下面给出对应点之间的映射公式：

- 就是前面每层都填充 $\left\lfloor \frac{k_i}{2} \right\rfloor$ 得到的简化公式： $p_i = s_i \times p_{i+1}$
- 需要把上面的公式进行级联得到： $p_0 = S \times p_{i+1}$ ，其中 $S = \prod_0^i s_i$
- 对于 feature map 上的 (x', y') ，它在原始图中的对应点为 $(x, y) = (Sx', Sy')$
- 论文最后的做法：把原始图片中的 ROI 映射为 feature map 中的映射区域（上图橙色区域）其中左上角取：

$$x' = \left\lfloor \frac{x}{S} \right\rfloor + 1$$

$$y' = \left\lfloor \frac{y}{S} \right\rfloor + 1$$

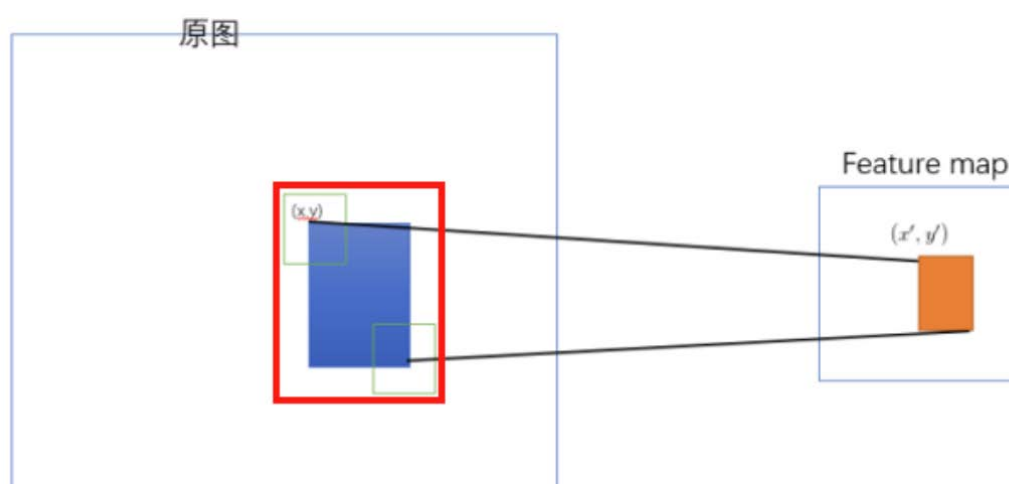
右下角的点取：

$$x' = \left\lceil \frac{x}{S} \right\rceil - 1$$

$$y' = \left\lceil \frac{y}{S} \right\rceil - 1$$

加 1 和减 1 的效果分别是增加和减少。也就是 左上角要向右下偏移，右下角要向左上偏移。

采取这样的策略是因为论文中的映射方法（左上右下映射）会导致 feature map 上的区域反映射回原始 ROI 时有多余的区域（下图左边红色框是比蓝色区域大的）



Given a window in the image domain, we project the left (top) boundary by: $x' = \lfloor x/S \rfloor + 1$ and the right (bottom) boundary $x' = \lceil x/S \rceil - 1$.

第二章：Fast R-CNN

Fast R-CNN 使用了深度卷积神经网络来对候选区域进行分类，并且与之前的工作相比，既能提高训练和测试时的速度，又能增加检测的精度。与 R-CNN 相比，训练时 Fast R-CNN 能快 9 倍，测试时能快 213 倍。与 SPP-Net 相比，训练时快

了 3 倍，测试时快了 10 倍。

在论文第一章的第一小节部分，分别介绍了 R-CNN 和 SPP-Net 的缺点。

首先看 R-CNN 的缺点：

- 需要针对大量的候选框分别进行计算
- 特征提取之后的分类器训练和位置回归，是几个独立步骤分别进行的

在训练过程中，提取的特征要先存储在硬盘上，然后训练 SVM 分类模型，最后训练位置回归模型，而测试过程也是类似的，特征提取之后，需要先进行 SVM 分类，再回归目标的准确位置，整个过程在计算时间和存储空间上，都需要很大的开销。

通过分析得知，R-CNN 之所以慢，它需要通过 CNN 前向传播来提取每一个候选区域的特征，没有用到共享计算。因此后续提出了 SPP-Net 网络。

SPP-Net 网络通过共享计算来加速 R-CNN 的训练和测试过程。核心是引入了 SPP 层，也叫空间金字塔池化层。

SPP-Net 的结构和 R-CNN 的结构大同小异，R-CNN 的框架里，全连接层之间是一个最大池化层，而在 SPP-Net 里面，全连接层前面的最大池化层换成了 SPP 层。

SPP 层的优点：当你传入一张图片时，通过卷积层和池化层，通过最后一个卷积层的时候，假如得到了一个 $13 \times 13 \times 256$ 的特征图。在 R-CNN 中，我们通常会使用一个 3×3 ，步长为 2 的最大池化层，把 $13 \times 13 \times 256$ 的特征图变成 $6 \times 6 \times 256$ 的特征图，把池化的结果拉平后传给全连接层。SPP 层本质上就是限制了我们的输出尺寸，也就是说采用了一个动态的池化核尺寸，比如我们想从 13×13 的特征图中得到 4×4 大小的特征图，那么我们只需要 $13/4$ 后向上取整来作为池化核的一

个尺寸， $13/4$ 向下取整的结果作为步长，也就是说用大小为 4，步长为 3 的池化核，最终得到 4×4 大小的特征图，接下来 2×2 和 1×1 的操作一样。

所以 SPP 层的池化核的尺寸和步长是动态变化的，它会根据你输出特征图的尺寸来动态的改变池化核的尺寸和步长。在得到 、 和 大小的特征图后，再把它们拼接起来，就形成了一个固定长度的特征，再把这个特征传给后续的全连接层。有了这个 SPP 层，可以输入任意大小的图片。

既然不需要像 R-CNN 一样限制输入图片大小的尺寸，那么换个思路，我们不传候选区域进去，直接把整幅图片传进 CNN 里面。SPP-Net 论文里面提到，原图中的候选区域位置和特征图上的候选区域位置对应的区域相同。因此可以直接在特征图上提取候选区域。虽然不同尺寸输入的图像的候选区域尺寸不同，但是可以利用 SPP 层将不同尺寸的特征图转换成一个特定大小的输出。

那么我们相当于只使用 CNN 进行了一次前向传播。也就是说我们实现了在整张图上进行了 **共享计算**。

SPP-Net 依旧存在很多问题，和 R-CNN 一样，训练依旧是一个多阶段的过程，最后利用全连接层提取到特征之后，还是和 R-CNN 一样，SVM 分类和边界框回归。既然是多阶段的，还是需要先将特征存储到磁盘里。R-CNN 的训练问题还是没有得到解决。

并且 SPP-Net 引入了新的问题：**微调算法很难去更新卷积层**。这句话的意思后面在细讲。

因此提出了 Fast R-CNN 来尝试解决 R-CNN 和 SPP-Net 的缺点。

Fast R-CNN 的优点如下：

- 检测质量(mAP)高于 R-CNN、SPPnet

- 训练是单阶段的，使用多任务损失
- 训练能更新所有的网络层
- 由于训练过程变成单阶段了，所以也就不需要将特征存储到磁盘中

Fast R-CNN 的模型结构和训练流程：

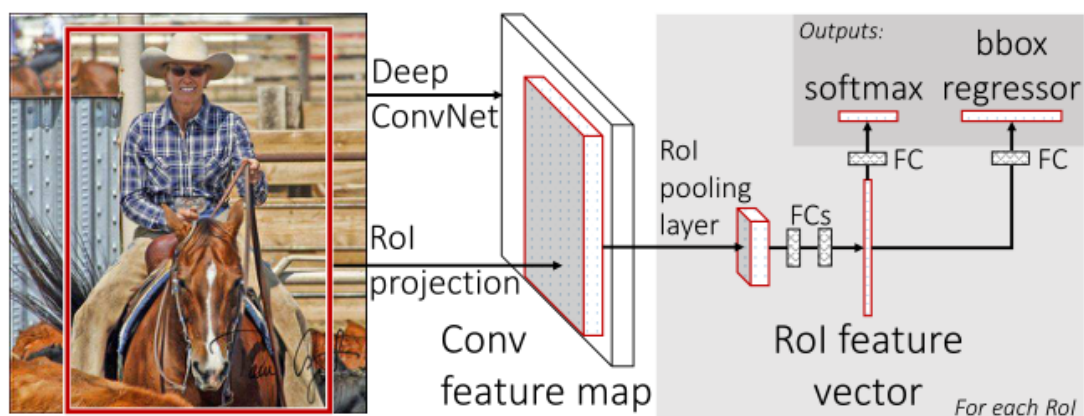


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax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

对于一整张输入图片，会先通过深度卷积神经网络来得到一个整张图片的图片特征，根据候选区域在原图上的一个图片位置，我们可以根据 ROI 映射来得到特征图上的候选区域位置。

有了候选区域的特征图之后，通过 ROI 池化层来把这个尺寸不固定的候选区域特征图转化成尺寸固定的候选区域的特征图。然后接上两个全连接层来得到每一个区域的特征向量，之后额外接上两个并行的全连接层，其中一个利用 softmax 分类 ($k+1$ 个类别)，另一个用坐标进行边界框回归 (输出 $[4 \times (k+1)]$ 个值)。

首先来看一下什么叫 ROI 池化层：

RoI pooling 借鉴了 SPPNet 的思想，是简化版的 SPPNet。它可以将不同大小的 feature map 转化为相同大小的特征向量。

相比于 SPPNet，RoI pooling 只进行了一组下采样。而 SPPNet 做了多组下采样，然后将结果拼接。RoI pooling 是将 SPPNet 的思想发挥到了精简的极致。对于一张 feature map，我们先提取出 region proposal 映射到的那部分 feature map 的区域。假如最终我们要生成 $H \times W$ 的 feature map，那么我们就将此 feature map 的区域划分为平均的 $H \times W$ 大小的网格。分别求出每个网格中的最大值，我们就会得到 $H \times W$ 大小的结果。这就是 RoI pooling 的过程。

【初始化预训练网络结构】

先看一下通过的网络模型（例如 AlexNet）：

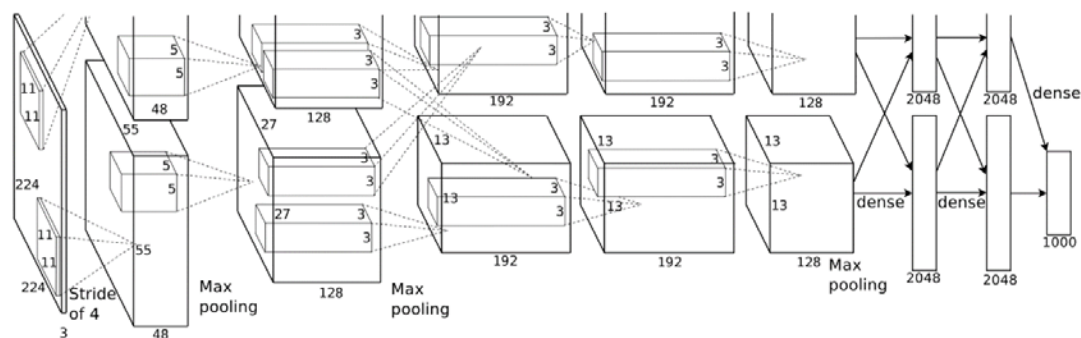


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–1000.

先将通用的网络模型（例如 AlexNet、VGG16 等）在 ImageNet 数据集上进行预训练。做了如下的 3 步变换：

- 将网络最后的 max pooling 层换成 RoI pooling 层
- 网络最后的 FC 层和 softmax 替换成并列的 2 层 FC 层。一个 FC 用来分类【输出长度为：类别数 + 1，即 21】，另一个 FC 用来回归最后的 Bounding Box。【输出长度为：4，通道数为：类别数 + 1】

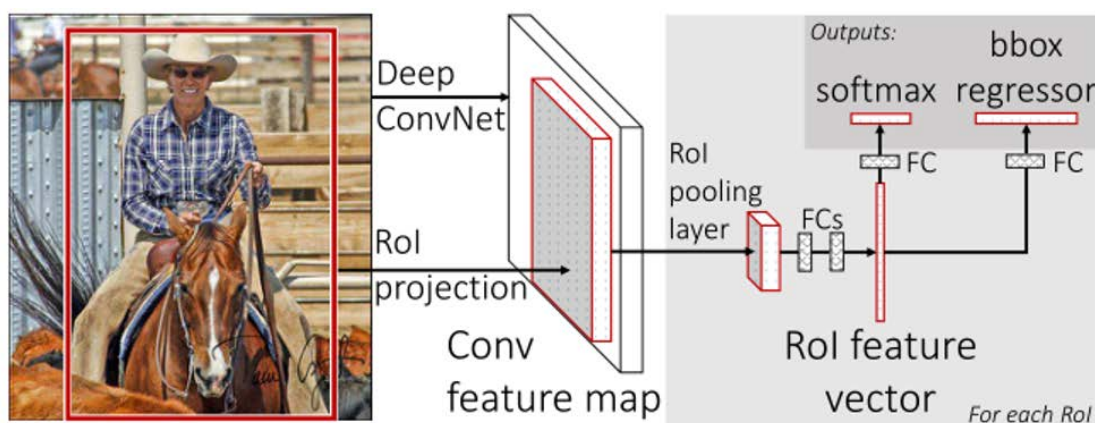
- 将网络修改为接收 2 个值为输入。图像的 list 和属于这些图像中的 roi 的 list。

【微调】

微调其实就是利用反向传播来训练模型的权重。Fast R-CNN 可以很好的利用反向传播去更新权重，而 SPP-Net 不可以。

严格说，SPP-Net 也可以反向传播，但是会复杂很多，所以用 cannot update the convolutional layers 其实是不算太准确的说法。Ross 大神在 Fast R-CNN 中给出的解释是：SPP 训练样本来自不同图像导致反向传播效率低下。我的理解是：SPP-Net 中 fine-tuning 的样本是来自所有图像的所有 RoI 打散后均匀采样的，即 RoI-centric sampling，这就导致 SGD 的每个 batch 的样本来自不同的图像，需要同时计算和存储这些图像的 Feature Map，过程变得 expensive。Fast R-CNN 采用分层采样思想，先采样出 N 张图像 (image-centric sampling)，在这 N 张图像中再采样出 R 个 RoI，具体到实际中， $N=2, R=128$ ，同一图像的 RoI 共享计算和内存，也就是只用计算和存储 2 张图像，消耗就大大减少了。

采样方法为什么这么重要？接下来展开讲一讲，先看一下流程图：



看一下模型结构，这里因为是测试一张图片的，所有只有一张图片的输入，但是在模型训练时，我们需要将一批一批的图片喂给卷积神经网络，一次会传入多张

图片。

在反向传播更新权重的时候,是需要利用我们正向传播得到的中间每一层的值去更新权重的。在 SPPNet 中,训练时输入的一批图片是通过随机采样得到的。比如说,我们需要 128 个候选区域,这 128 个区域可能是来自不同的图片,也就是说我们需要传入 128 张这样类似的原图进去,然后在 128 张特征图中各自扣除需要的候选区域。在前向传播和后向传播的过程中,会占用大量的显存,降低模型的训练效率。

我们需要解决 SPPNet 不能微调 SPP Layer 之前的卷积层的问题。

设 N 为图像的 batch 的个数。 R 为一个 batch 的总 ROI 的个数。

使用分层采样。先采样 N 张图片组成一个 list,再对每个图片采样 $\frac{R}{N}$ 个放入 list。

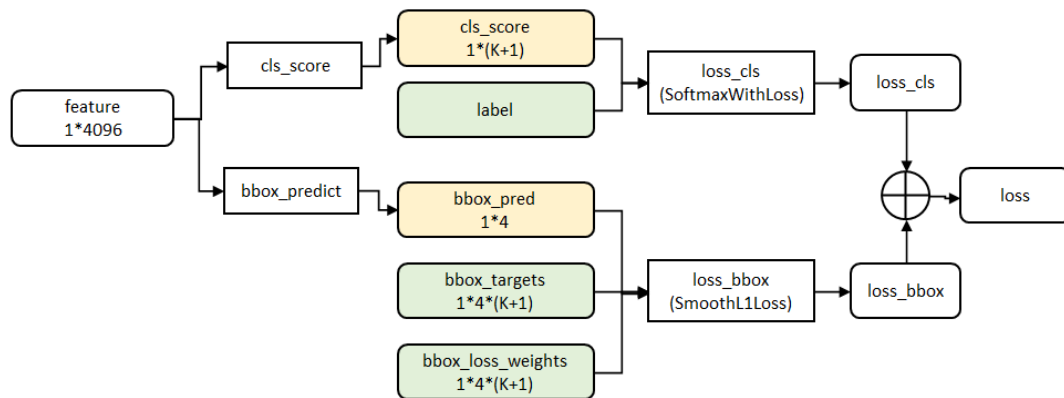
这样的话我们每次从 ROI 的 list 取出 $\frac{R}{N}$ 个,就是对应的那个图片生成的那些 ROI。

这样的策略,可能会导致收敛缓慢。因为同一个样本的 ROI 有关联性。不过作者在试验中发现,在实际中并没有出现这种问题。作者使用的 $N=2, R=128$ 。

在前向传播的时候,我们只需要传两张图片进去,因为每张图片的 64 个区域都是在一张图上得到的。也就是说我们两张原图传进去,然后在其特征图上各自扣出 64 个候选区域即可,而不是像 SPPNet 一样,需要去传入 128 个原图,大大提升了训练效率,这就是作者提出的为什么 SPPNet 无法更新卷积层权重的原因。

因为效率太低了!

【多任务损失函数】



在 R-CNN 中的流程是先提 proposal, 然后 CNN 提取特征, 之后用 SVM 分类器, 最后再做 bbox regression 进行候选框的微调; Fast R-CNN 则是将候选框目标分类与 bbox regression 并列放入全连接层, 形成一个 multi-task 模型。

- cls_score: 用于分类, 输出 $k+1$ 维数组 p 。表示属于 k 类物体和背景的概率;
- bbox_predict: 用于调整候选区域位置, 输出 $4 \times k$ 维数组, 也就是说对于每个类别都会训练一个单独的回归器。
- loss_cls: 评估分类代价, 由真实分类 μ 对应的概率决定: 用 $L_{cls}(p, u) = -\log(p_u)$ 表示。
- loss_bbox: 评估回归损失代价, 比较真实分类 μ 对应的预测平移缩放参数

$t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$ 和真实平移缩放参数 $v = (v_x, v_y, v_w, v_h)$ 的差距, 用 smooth L1 表示:

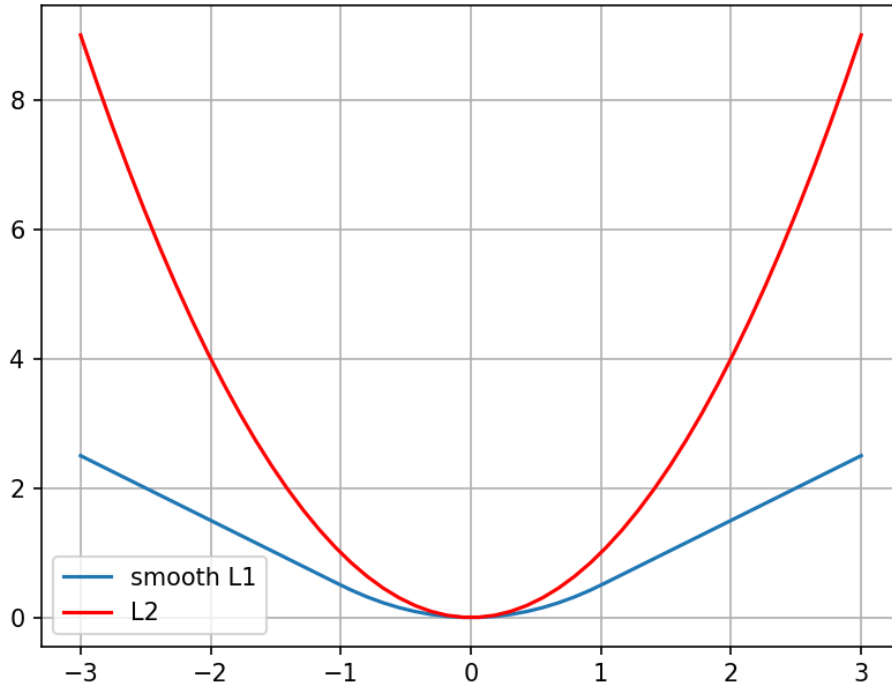
$$L_{loc}(t^u, v) = \sum_{i \in \{x, y, w, h\}} \text{smooth}_{L_1}(t_i^u, v_i)$$

其中:

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & |x| < 1 \\ |x| - 0.5 & \text{other} \end{cases}$$

R-CNN 中使用的 L2 损失函数为: $L_2(x) = x^2$

看一下二者之间的图像:



如果我们使用 L_2 损失，那么随着目标值和预测值之间的偏差不断增大， x 也不断增大，损失函数的梯度不断增大，可能引起梯度爆炸，无法进行训练。Smooth L1 在 $|x| > 1$ 的时候写成 $|x| - 0.5$ 的形式，保证梯度恒等于 1，不会产生梯度爆炸。选择系数为 0.5 的原因是使得 x 在正负 1 的位置梯度是连续的。

结合分类损失和回归损失，Fast R-CNN 微调阶段总的损失函数为：

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v)$$

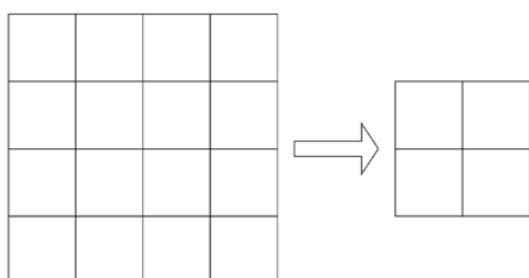
其中：

$$[u \geq 1] = \begin{cases} 1 & u > 1 \\ 0 & otherwise \end{cases}$$

约定 $u = 0$ 为背景分类，那么 $[u \geq 1]$ 函数表示背景候选区域。即负样本不参与回归损失，不需要对其进行回归操作。 λ 控制分类损失和回归损失的平衡，文中所有实验 $\lambda = 1$ ，即 L_{cls} 和 L_{loc} 按照等比例相加。

【池化层（pooling）的反向传播是怎么实现的】

卷积神经网络中一个不可导的环节就是 Pooling 池化操作，因为 Pooling 池化操作使得 feature map 的尺寸发生变化，假如做 2×2 的池化，步距为 2，假设第 $l+1$ 层有 4 个梯度，那么第 l 层就会有 16 个梯度，这使得梯度无法对位的进行传播下去。

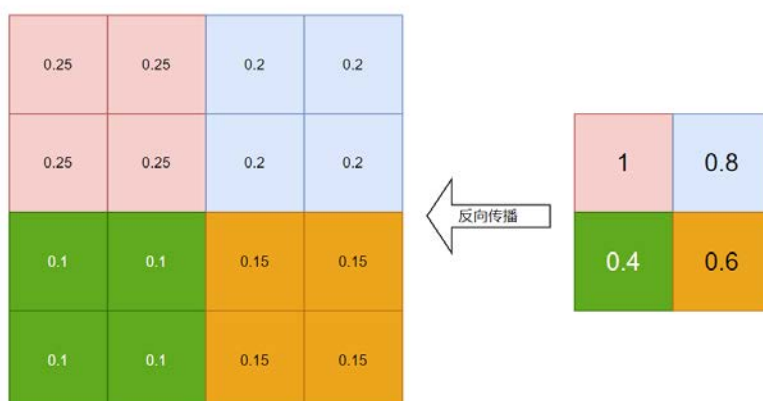


其实解决这个问题的思想也很简单，就是把 1 个像素的梯度传递给 4 个像素。但是需要保证传递的 loss（或者梯度）总和不变。

根据这条原则，mean pooling 和 max pooling 的反向传播也是不同的。

● mean pooling

mean pooling 的前向传播就是把一个 patch 中的值求取平均来做 pooling，那么反向传播的过程也就是把某个元素的梯度等分为 n 份分配给前一层，这样就保证池化前后的梯度（残差）之和保持不变，还是比较好理解的，图示如下：

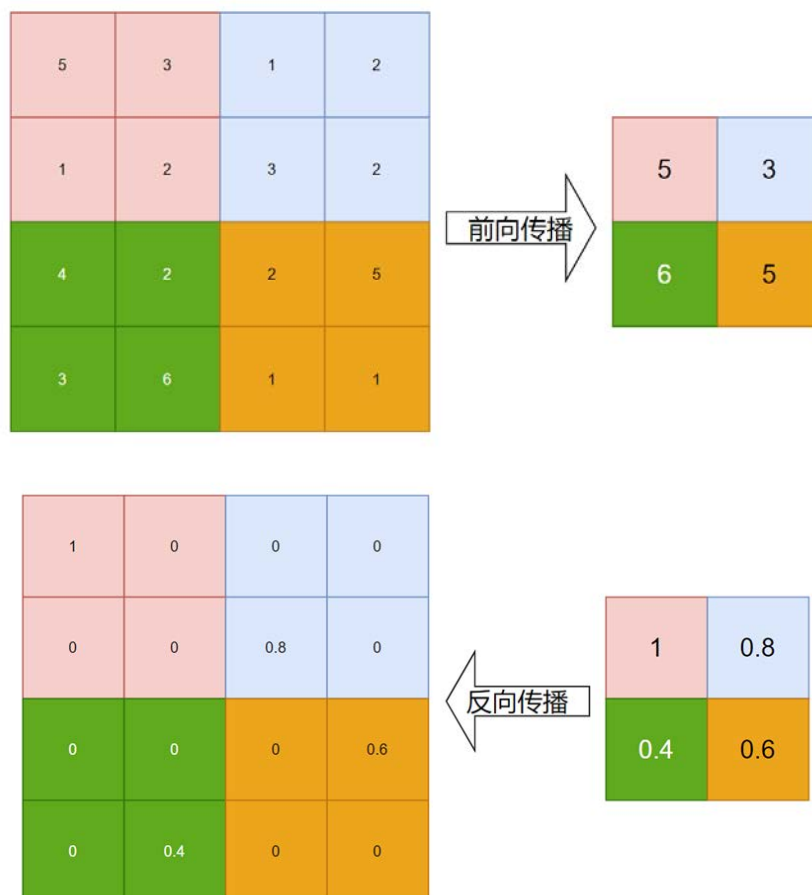


这里有一个容易犯错的地方，不是简单的把梯度复制 N 遍后直接反向传播回去，

这样会造成 loss 之和变为原来的 N 倍，网络是会产生梯度爆炸的！

- max pooling

max pooling 也要满足梯度之和不变的原则，max pooling 的前向传播是把 patch 中最大的值传递给后一层，而其他像素的值直接被舍弃掉，那么反向传播也就是：



所以反向传播也就是把梯度直接传给前一层某一个像素，而其他像素不接受梯度，也就是为 0。所以 max pooling 操作和 mean pooling 操作不同点在于需要记录下池化操作时到底那个像素的值是最大，也就是 max id，这个变量就是记录最大值所在的位置，因为在反向传播中要用到。

总结一下：

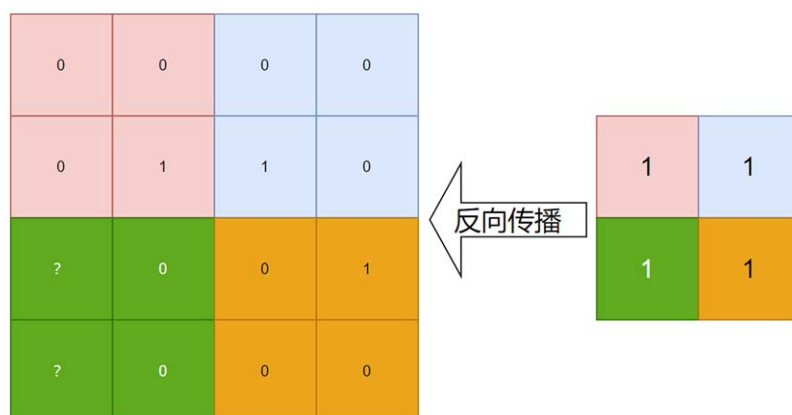
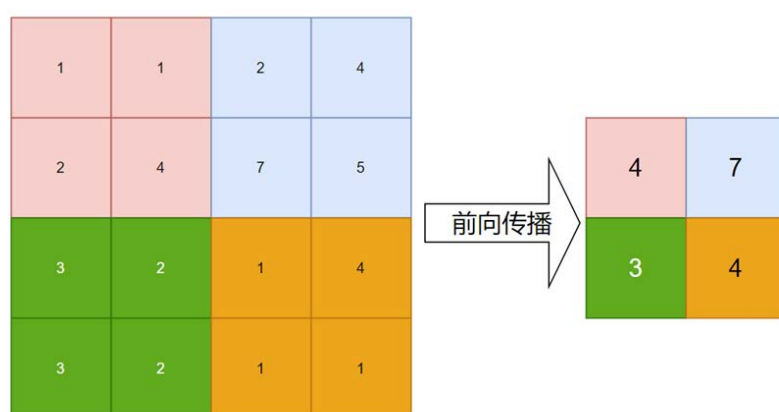
x_i 为输入层节点， y_i 为输出层节点，那么损失函数 L 对输入层节点 x_i 的梯度为：

$$\frac{\partial L}{\partial x_i} = \begin{cases} 0 & \delta(i, j) = false \\ \frac{\partial L}{\partial y_j} & \delta(i, j) = true \end{cases}$$

其中判决函数 $\delta(i, j)$ 表示输入 i 节点是否被输出 j 节点选为最大值输出。

不被选中 ($\delta(i, j) = false$) 有两种可能: x_i 不在 y_i 范围内, 或者 x_i 不是最大值。

若选中 ($\delta(i, j) = true$) 则由链式规则可知损失函数 L 相对 x_i 的梯度等于损失函数 L 相对于 y_i 的梯度 (为 1), 从而可得上述公式。

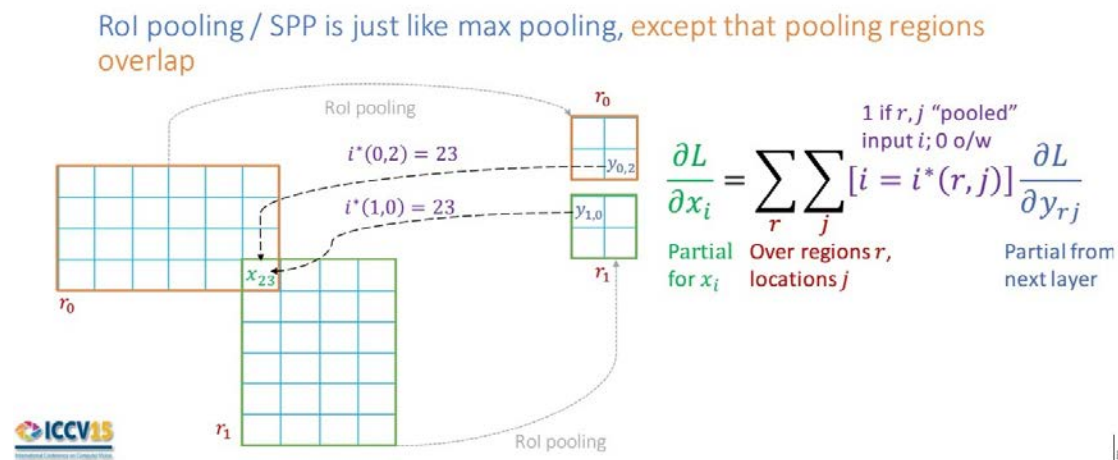


这里有一个疑问, 如果两处都是 3, 该怎么进行选择?

【目标检测算法中 ROI 层的反向传播】

和池化层的误差反向传播类似, 由于通过 SS 算法获得候选框映射到特征图后发现很多区域会有重合, 也就是说前一层的节点误差对后一层多个输出节点的误差有贡献, 因此, 前一层的误差应该等于后一层多个相关节点误差反向传播后的取

值之和。



为了便于推导，我们这里只选择一张图，即 mini-batch 的 N 为 1，此时有 128 张候选区域通过池化层。

设 x_i 是 ROI 池化层的第 i 激活输入， y_{rj} 是第 r 个 ROI 层的第 j 个输出。ROI 池化层计算 $y_{rj} = x_{i^*(r,j)}$ ，其中 $i^*(r, j) = \arg \max_{i \in R(r,j)} x_i$ 。 $R(r, j)$ 是在输出单元 y_{rj} 子窗口上的一组输入索引，就是格点区域的所有索引。一个 x_i 可以被分配几个不同的输出 y_{rj} 。ROI 池化层的反向传播函数计算是每个输入变量 x_i 损失函数的偏导数，如下所示：

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r, j)] \frac{\partial L}{\partial y_{rj}}$$

总之，对于每个批量中的 ROI 的 r 和对于每个池化输出单元 y_{rj} ，如果 i 是 y_{rj} 最大池化的最大选择，则加上所有相关的偏导数 $\frac{\partial L}{\partial y_{rj}}$ 来求得 $\frac{\partial L}{\partial x_i}$ ，也就是说，这些 y_{rj} 由相关 x_i 贡献得到，再反过来求 x_i 对于 L 的偏导数。

【SGD 超参数选择】

除了修改增加的层，原有的层参数已经通过预训练方式初始化：

用于分类的全连接层以均值为 0、标准差为 0.01 的高斯分布初始化。

用于回归的全连接层以均值为 0、标准差为 0.01 的高斯分布初始化，偏置都初始

化为 0。

针对 PASCAL VOC 2007 和 2012 训练集，前 30K 次迭代全局学习率为 0.01，每层权重学习率为 1 倍，偏置学习率为 2 倍，后 10K 次迭代全局学习率更新为 0.0001。

动量设置为 0.9，权重衰减设置为 0.0005。

【尺度不变性】

作者提出了使用两种方式对规模不变的对象进行检测：brute-force（单一尺度）和 image pyramids（多尺度，图像金字塔）。

单一尺度直接在训练和测试阶段将 image 预先固定好像素大小，直接输入网络训练就好，然后期望在训练过程中网络自己能够学习到尺度不变性 scale-invariance。

多尺度在训练阶段随机从图像金字塔（缩放图片的 scale 得到，得到多尺度图片，相当于扩充数据集）中采样训练，通过一个图像金字塔向网络提供一个近似的尺度不变，在测试阶段图像金字塔用来对每个候选框近似尺度归一化，训练阶段每次采样一个图像就随机采样一个金字塔尺度。

作者在 5.2 节对单一尺度和多尺度分别进行了实验，不管哪种方式下都定义图像短边像素为 s ，单一尺度下 $s=600$ 【维持长宽比进行缩放】，长边限制为 1000 像素；多尺度 $s=\{480,576,688,864,1200\}$ 【维持长宽比进行缩放】，长边限制为 2000 像素，生成图像金字塔进行训练测试；实验结果表明 AlexNet 【S for small】、VGG_CNN_M_1024 【M for medium】下单一尺度比多尺度 mAP 差 1.2%~1.5%，但测试时间上却快不少，VGG-16 【L for large】下仅单一尺度就达到了 66.9%的 mAP 【由于 GPU 显存限制多尺度无法实现】，该实验证明了深度神经网络善于直

接学习尺度不变形，对目标的 scale 不敏感。

第 2 中方法的表现确实比 1 好，但是好的不算太多，大概是 1 个 mAP 左右，但是时间要慢不少，所以作者实际采用的是第一个策略，也就是 single scale。

【Fast R-CNN detection】

一旦 Fast R-CNN 网络被微调，检测相当于运行正向传播（假设对象建议框 object proposal 是预先计算的）。

网络将图像（或图像金字塔，编码为图像列表）和待给得分的 R 对象建议框（object proposal）列表作为输入。

在测试阶段，R 大约为 2K 个，当使用图像金字塔的时候，每个 RoI 被指定尺度使得接近 224*224。对于每个测试 RoI r ，网络输出关于 r 的一个后验概率分布 p 和一系列预测 bbox 偏移（每个类 [共 k 个类] 获得自己的精确 bbox 预测）。

然后使用估计概率 $\Pr(class = k | r) = p_k^\Delta$ 给 r 赋予关于 k 个对象类的检测置信度。最后给每个类都实施一个非极大值抑制。

【SVD 全连接层加速网络】

图像分类任务中，用于卷积层计算的时间比用于全连接层计算的时间多，而在目标检测任务中，selective search 算法提取的建议框比较多【约 2k】，几乎有一半的前向计算时间被花费于全连接层，就 Fast R-CNN 而言，RoI 池化层后的全连接层需要进行约 2k 次【每个建议框都要计算】，因此在 Fast R-CNN 中可以采用 SVD 分解加速全连接层计算,具体实现如下：

1. 物体分类和窗口回归都是通过全连接层实现的，假设全连接层输入数据为 x ，

输出数据为 y ，全连接层参数为 W ，尺寸为 $u \times v$ ，那么该层全连接计算为：

$$y = Wx \quad (\text{计算复杂度为 } u \times v)$$

2. 若将 W 进行SVD分解,并用前 t 个特征值近似代替,即:

$$W = U \Sigma V^T \approx U(u, 1:t) \Sigma(1:t, 1:t) \cdot V(v, 1:t)^T$$

那么原来的前向传播分解成两步:

$$y = Wx = U(\Sigma V^T) \cdot x = U \cdot z$$

计算复杂度为: $u \times t + v \times t$, 若 $t < \min(u, v)$, 则这种分解会大大减少计算量。

在实现时,相当于把一个全连接层拆分为两个全连接层,第一个全连接层不含偏置,第二个全连接层含偏置。实验表明, SVD 分解全连接层能使 mAP 只下降 0.3% 的情况下提升 30% 的速度,同时该方法也不必再执行额外的微调操作。(见下图):

