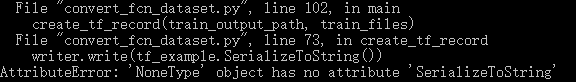
首先是生成TF\_record文件，下载了pascal\_voc数据集，其中原图和ground truth都是图片的形式，而不像目标检测中的标签那样包含名称，类别、bounding-box参数等的xml文件，故本次采用将原图和对应的ground truth通过zip函数生成一一对应的元组，然后在生成字典的时候一组一组地调用出来，过程中出了两个问题，一个是



是因为直接按照目标检测的方式写了，回过头来一想，目标检测的data指的是每个解析后的xml文件，这里直接去掉filename就可以了，因为data就是文件名。

另一个问题：



意思是属性异常，结果是None，所以也就没有SerializeToString方法，自然就不能调用了。解决办法：在前面加个if条件判断是否是None，是None就continue，跳过下面的写入操作。

然后将生成的tf\_record文件和在github/tensorflow/models/slim下找到的VGG\_16.ckpt一起传入数据集。

在训练文件中调用VGG网络最后的logits并设置fc6即全卷积层的padding=SAME，于是得到的是7X7的尺寸，然后经过2倍上采样再与VGG网络pool4的logits相加，所得结果再2倍上采样后和VGG网络pool3的logits相加，此时只需要上采样放大八倍即可还原为原图尺寸，故也称为fcn\_8s。

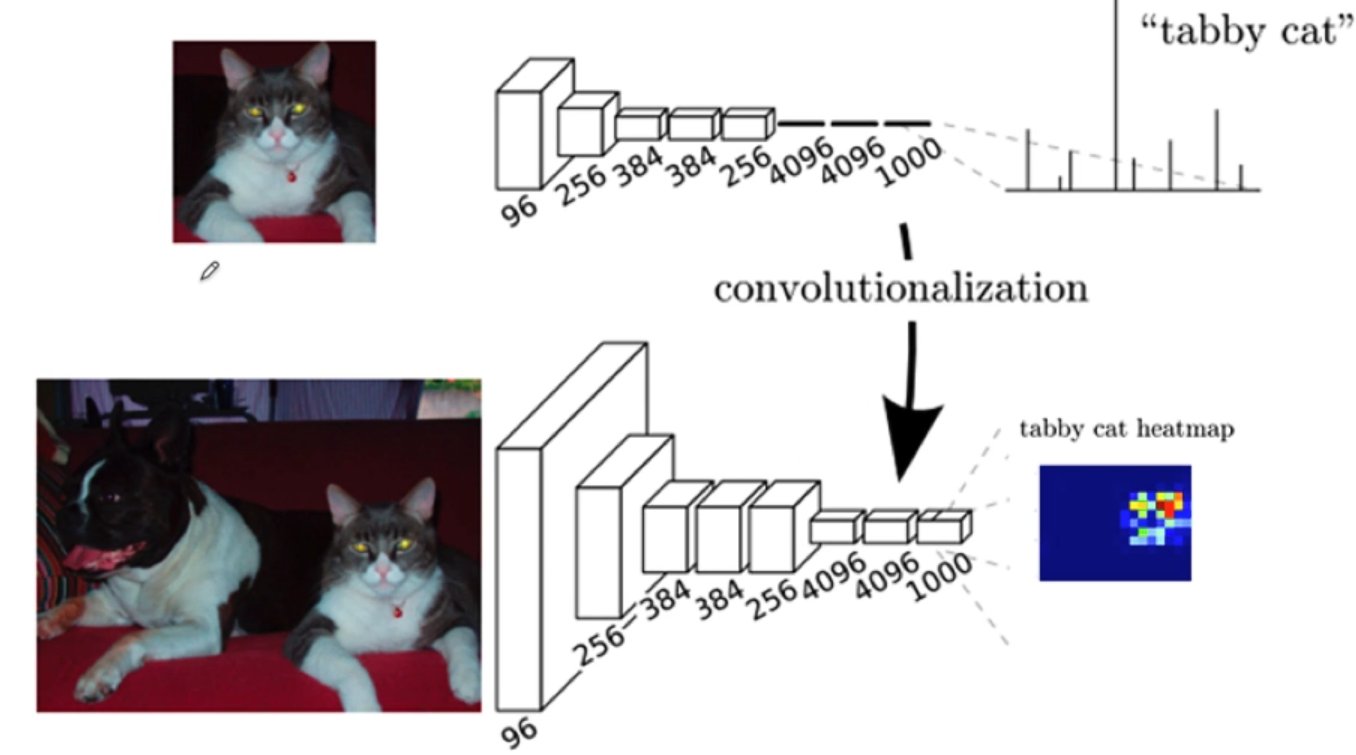
图中分别为原图、预测的分割结果图、预测加了CRF后端处理的结果图

**语义分割：**

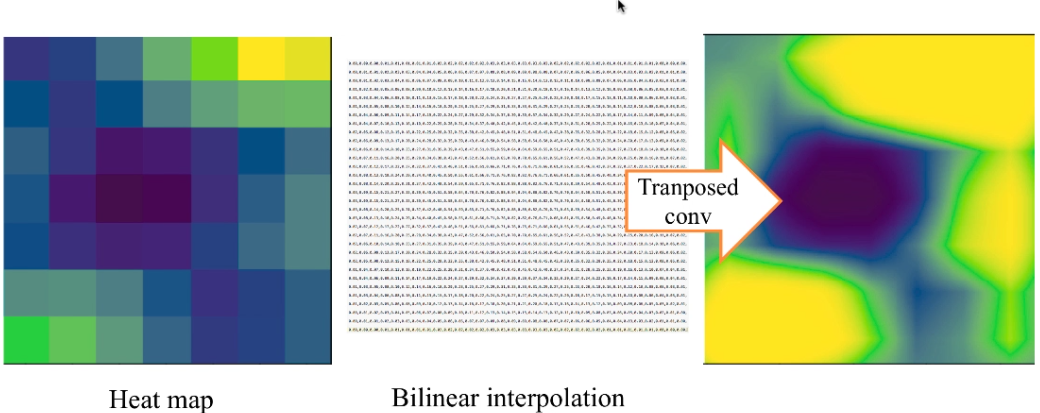
**FCN（Fully Convolutional Networks for Semantic Segmentation）:**

特点：全卷积，把所有全连接结构用全卷积代替，如输入为3X3，kernel也为3X3,相当于两个矩阵的element\_wise相乘再全部相加；多层feature跳接结构

在最后提取的特征图后，不是像目标检测一样去生成对应目标bounding-box,而是去产生它的分割结果，那么就要用到反卷积操作，即以双线性插值方式初始化一个kernel,再进行反卷积。



如图，输入一张被裁切为224X224的图片得到一维的向量，最后得到一个分类的结果（长度为1000以one-hot的形式），当把它改造成了全卷积网络之后，就可以输入任意大小的图片，输出的长和宽空间尺度、比例和原图是一致的，但是尺寸比原来缩小1/32。比如说我们从1000张feature map中挑出tabby cat所对应的分类，图上为tabby cat heatmap，它是一个概率的表示，代表着这张图每一个像素点都映射到原图的某一个区域上，它的值代表着原图中所对应的区域是tabby cat这个分类的概率的高低。这样我们就能做一个比较粗糙的分割，即取概率大于一定阈值的feature map映射到原图，大概可以得到原图中猫的区域，但是我们要精确到原图的每一个像素（轮廓），那么就需要反卷积。

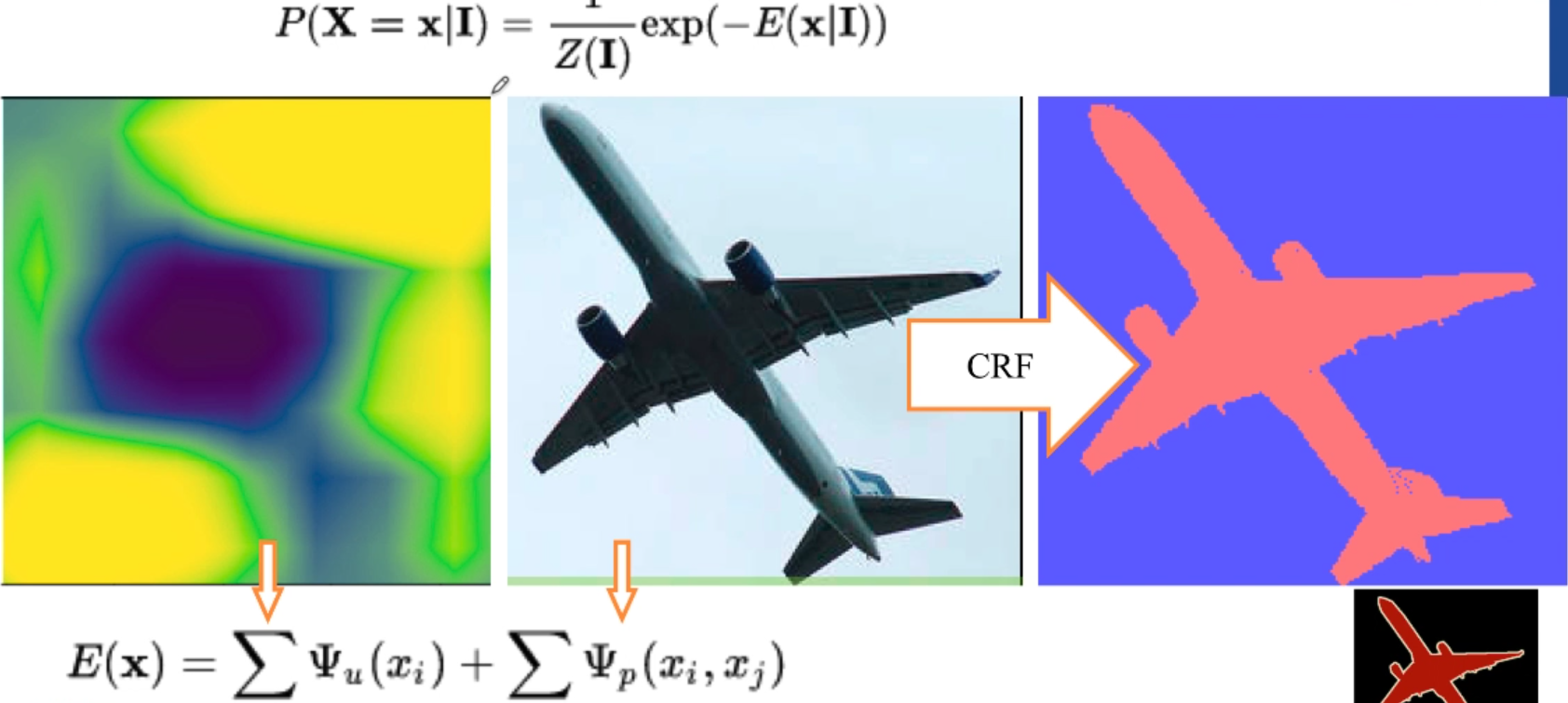


我们知道VGG网络最后输出的特征图在空间尺度上为原图的1/32，所以这里施加一个32的stride，用一个64的kernel对它进行反卷积，相当于把它在空间尺度上扩大了32倍，可以得到一个和原图等大的还原后的feature map。FCN使用的是32X32的双线性插值。

那么为什么不是训练完成之后再利用双线性插值对结果进行优化呢？是因为ground truth是和原图等大的一个像素级别，我们要进行优化有两个办法，一个是把ground truth进行降采样成7X7大小，另一个就是把Heat map上采样得到一个和原图等大的图像然后和ground truth进行比较。可想而知，降采样得到的结果会更加不精确，所以把双线性插值作为卷积过程中的一部分，采用反卷积来实现它，作为我们训练过程中可以进行反向传播的一个步骤，而不是直接进行双线性插值。

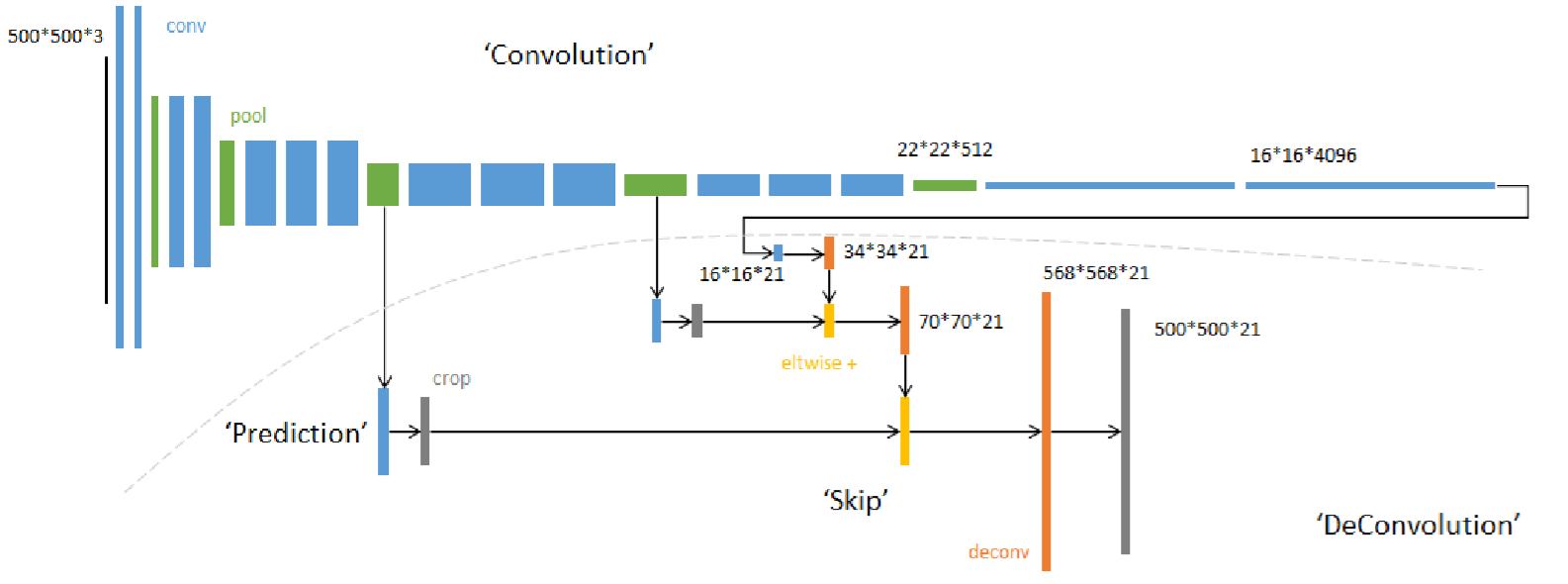
总结：反卷积的目的就是把得到的比较小的feature map上的概率分布变成和原图一样大的分布，使用反卷积的意义在于它能够进行反向传播并且把输入从空间尺度上进行扩大，而在反卷积中使用的kernel就是一个双线性插值。

仅仅由双线性插值得到的这样的图片还不够，还要进行后端处理CRF(条件随机场)，就算CRF时将原图也作为了输入加入计算，即原图中任意两个像素属于同一个物体的概率，两个像素间包含颜色、亮度等信息，由这些信息计算它们关系。



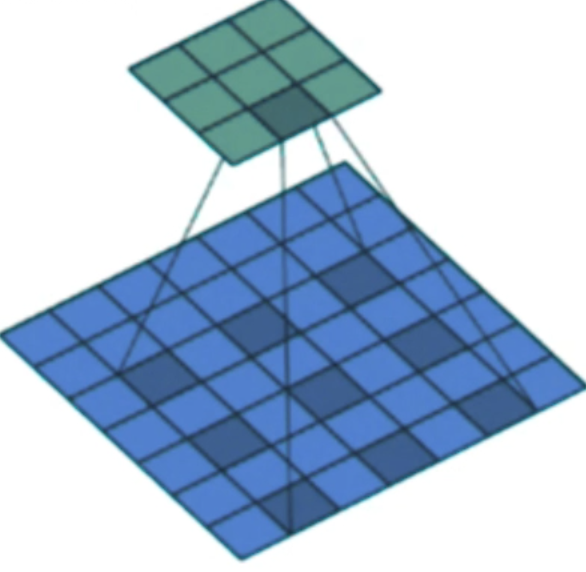
多层跳接结构：

由于反卷积后得到的结果仍然太过粗糙，原因是因为整个网络在最后一步的时候它的空间尺度是原来的1/32，我们把它进行双线性插值得到和原图等大的一个分布图，但是双线性插值毕竟是一个比较粗糙的比较机械化的方法，从根本上解决这个问题的办法就是，让我们的结果的结果变得更密集一些，fcn中做了很多工作，比如说去掉pooling层，但是去掉后，它后面层的feature map映射到原图的感受野是缩小的，而造成的结果并不是没法训练，问题是如果我们采用pre-train的模型的话，它原来的权重是在原有的感受野的大小下进行训练的，如果去掉pooling或者stride改成1，它的感受野尺寸变小了，那么这些权重也不再适用，相当于要重新对权重进行调优。于是就得出了多层跳接结构：



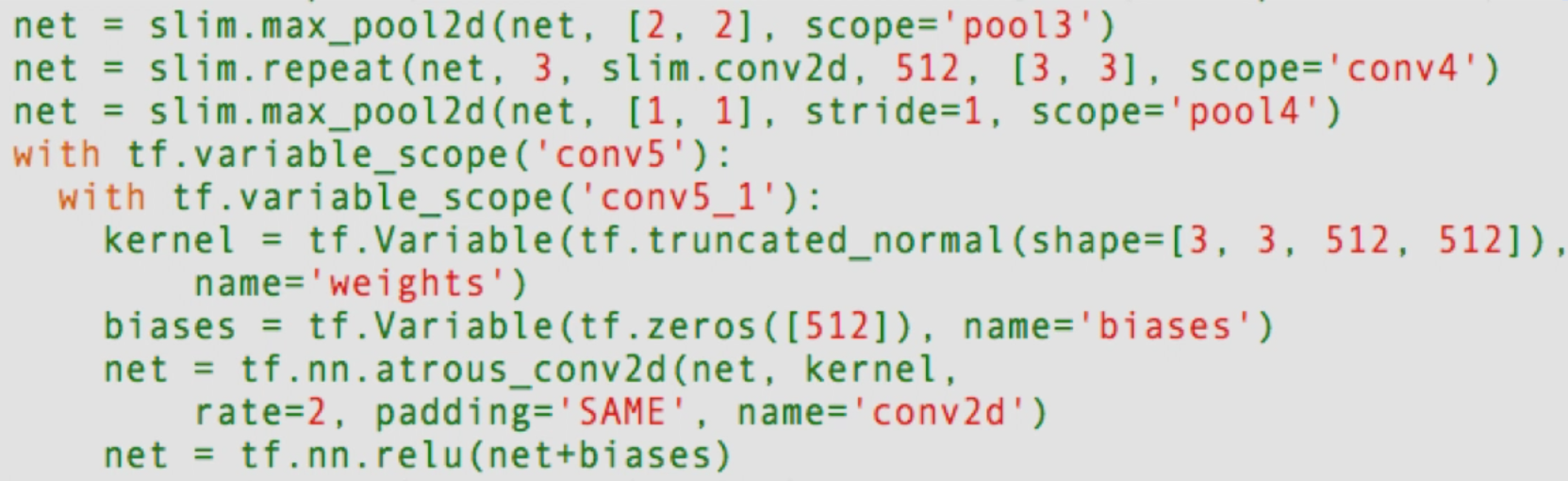
即对最后一层的输出进行2倍的上采样，再和前面某层进行element\_wise相加，得出的logits再进行2倍的上采样然后再与更前面的某层进行elemet\_wise相加（这个时候密度已经达到了原来的4X4=16倍），最后再反卷积成原图的大小。

**A’trous conv/Dilated conv/多孔卷积：**



如图，利用一个3X3的卷积获得5X5的感受野，多孔卷积的第一个效果就是扩张感受野的范围，第二个效果就是缩小了获得的feature map的尺度。它的效果和pooling、池化是类似的，起到了降维的效果，也起到了增加感受野的效果。

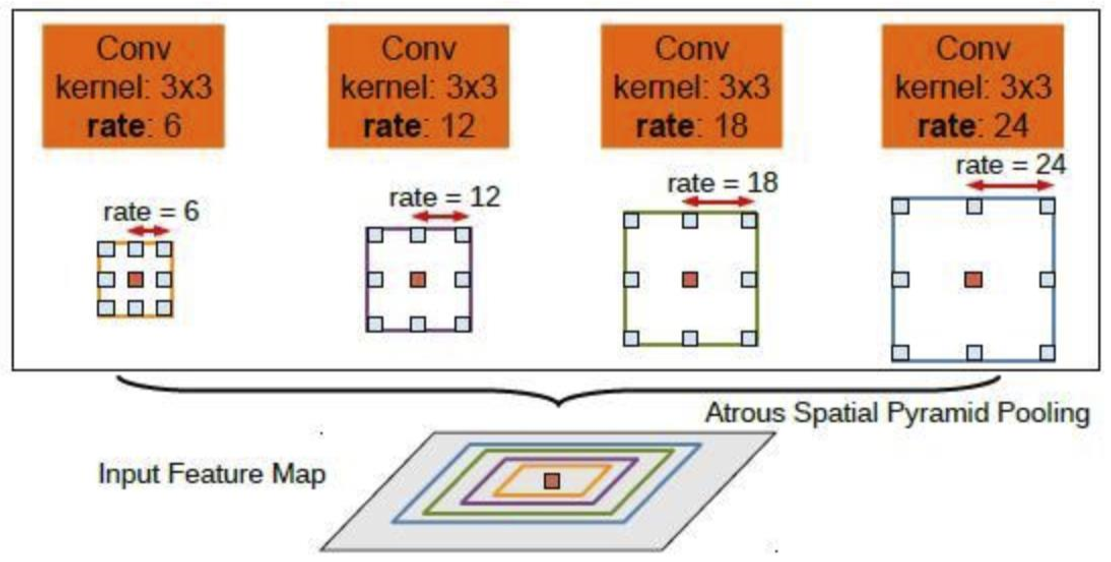
为什么会在segmentation提到利用多孔卷积呢？我们得到的预测的结果太过于粗糙，是因为我们在不断地池化，连续地进行池化，最终把空间尺度缩减为原来的1/32，我们其实希望得到一个更密集的结果。Fcn并没有办法解决，比如我们把pooling5改为stride=1，那么按照原来Fcn的方式后面所有的权重都会因为感受野的改变变得不再适用，训练需要花很长的时间才能收敛。如果我们把pooling5后面的卷积改为多孔卷积，那么感受野会再次被扩张到原来那么大，权重就不需要更多的预训练，和不再适用相比，重新训练的难度要小很多，甚至仍然可以使用原来的权重获得一个接近最优的结果。



如图，采用stride=1,那么输出的长和宽各自增长一倍，多孔卷积采用Same padding,就回避了多孔卷积会缩小一倍尺寸的问题，而感受野和VGG是同等大小，这样就能完美地利用VGG的权重。并且，如果我们只改一个pooling,如pool4,则长和宽各自扩张了两倍，如果把后面的pooling5也同时改掉，相当于长和宽各自扩大了四倍，就能起到和fcn\_8s同样的效果，而又不要像fcn\_8s那样额外地训练两个分类器用于最后地叠加，也不需要多次进行上采样。我们只需要把卷积改成多孔卷积，可以说是fcn\_8s的廉价版，但是用更少的时间，更少的迭代次数，运行效率更高。

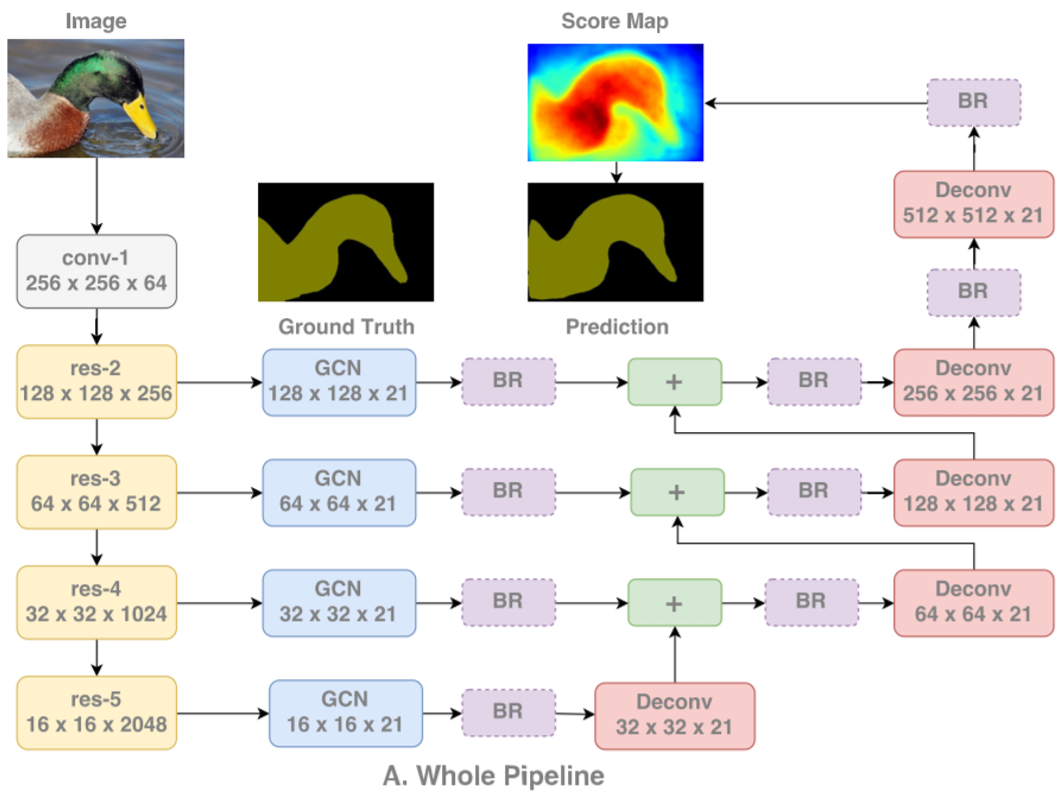
**Deep lab v2:**

在上一个算法的基础上，即两个池化stride=1，然后用多孔卷积替换掉后面的卷积。修改原来的全连接层，即我们用7X7去卷积得到一个4096维向量的层。把7X7的kernel都变成了3X3，然后像inception网络那样在输入的地方开了四个branch,每个branch输出1024长度的feature maps，四个加起来依然是4096。但这四个并不是做完3X3的卷积生成1024再加起来，按原来的方式继续1X1的卷积然后做分类，而是每一个branch一直跑到分类的阶段，在分类的结果层面上把结果拼合起来，四个branch唯一的区别在于rate,即孔的大小是不一样的，最后的结果是每一个branch映射到原图的感受野范围各自不同。那么，就实现了一个多层级的在不同的空间感受野的层面上对信息的捕捉，最后再把它组合起来。

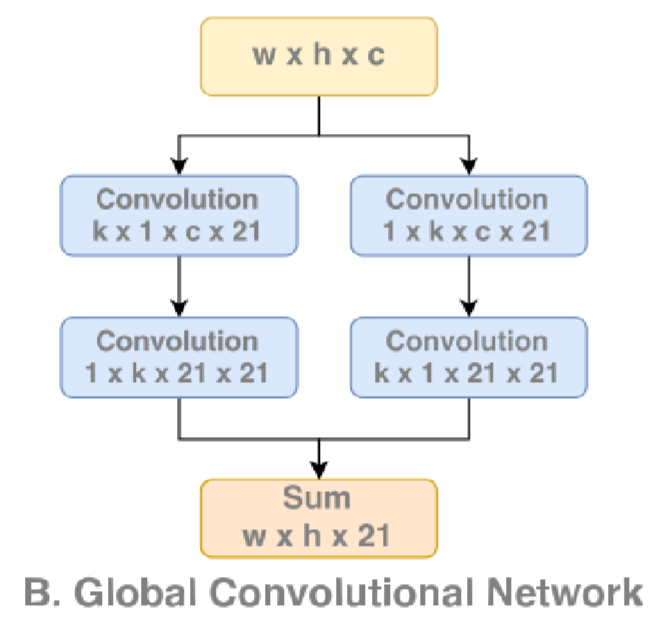
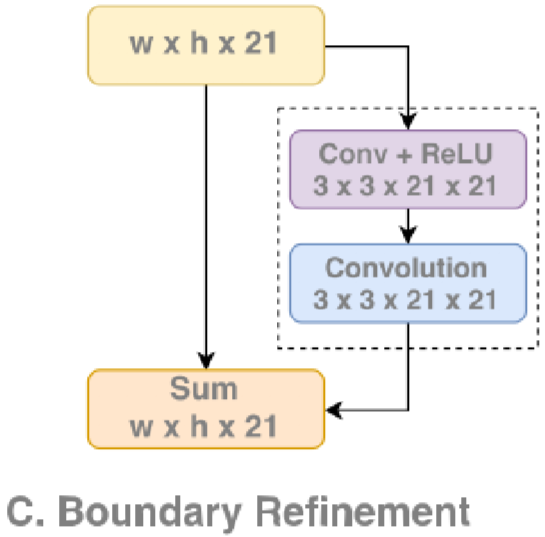


**GCN(Global Convolutional Network):**

主要思想是大内核（Large Kernel），采用一个非常大的卷积核对图片进行卷积，起到了一个全局卷积的作用。



在基础网络部分（卷积）resnet的不同block上直接进行输出，输出到GCN单元中，由于kernel(比如第一个)是128X128X21,计算量太大，所以在GCN单元中采用inception中常用的结构，即首先拆分成两个branch，每个branch又把非常大的kernel拆分成1Xk和kX1；从GCN出来后再经过BR模块（边缘增强）.

残差网络的每一个block都经过GCN和BR,输出也经过一个反卷积之后和前一个block的结果相加，然后再经过一个BR，再进行上采样，再和上一个的结果相加，这样一个堆叠循环的结构最后得到一个输出（Score Map）

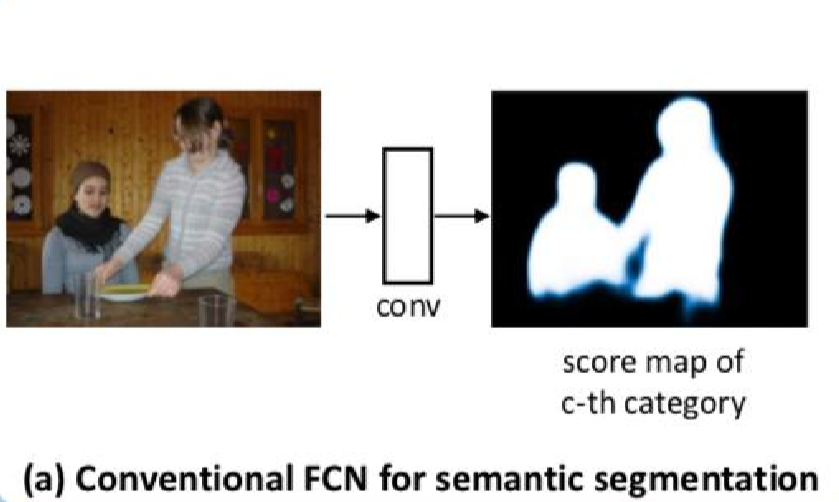
**Deep lab V3:**

创新点：增加了空间金字塔(Spatial Pyramid Pooling),在其中加入了Batch Normal，并且证明能对结果有不错的性能提升，此外，把CRF这种后端从网络中移除了，依然能得到一个好的效果。

**实例分割**：

**FCIS(Fully Convolutional Instance-aware Semantic segmentation):**

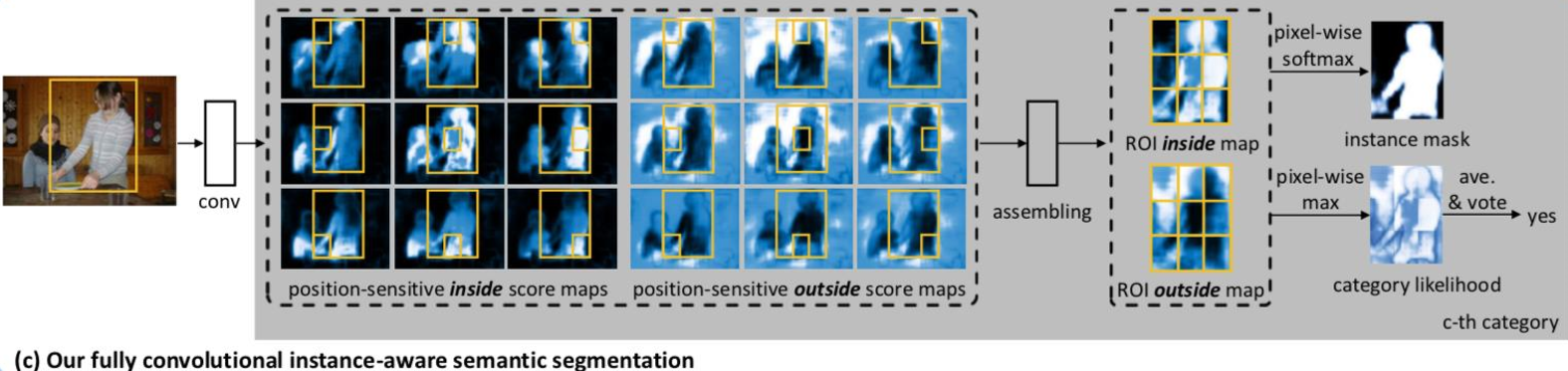
它融合了很多检测的思想，达到了端到端的分割，按之前的工作来看：



fcn并没有达到实例级别的效果。之前一直都采用的是分布训练，即检测归检测，分割归分割，检测出了结果我们再对输出的bounding-box做fcn。最大的问题在于，如下图：



框中不止包含大人，小孩也有一部分也包含进去了，如果做fcn，小孩的右半身也分到了大人这边去。InstanceFCN把框区分成了3X3的小块，然后对每个小块判定它属于目标物体的左侧还是右侧，会给出打分。图中小孩在人的右边，但是判断小孩的像素是属于人物的左边，所以产生的冲突，那么在标定的时候就会标定属于人物右手边的像素，实际上是个非常繁杂的工作，也充满的工程上trick这样的手段，但是结果还不错。



而在FCIS里面是判定两个分数，即在某个小方块里是insight还是outsight,不需要判断是在物体的左边还是物体的右边，只需要给出某一个区域内，小方块里面是属于bounding-box内部还是外部。同时结合了mask和分类，输出掩码和分类这两个任务同时训练。作者采用resnet作为基础网络，并进行很多修改，如多孔卷积，修改层次的深度。

**Mask R-CNN:**

结构上为Faster R-CNN+FCN，后端根据RPN给出的候选区域，在feature map上把候选区域的特征图切出来，把不同形状的region整理成同一个形状进行处理，然后计算它的分类、回归位置。我们所知道的Faster R-CNN同时进行两个任务，一个是进行候选区域的分类，另一个是对候选框位置的优化，Mask R-CNN是在两个任务的基础上加入了一个像素级别的物体分割，也就是对mask进行回归的任务。这篇论文所特有的思想是RoI Align，即采用双线性插值的方式，允许非整数像素的切割点；保留浮点数，先记录下相应的带浮点值得坐标，不进行相应的对齐，然后在Align的时候也记录下它应该分成多少份，以及它在feature map上的坐标，真正进行截取的时候采用双线性插值把feature map进行内部的插值然后在合适的位置上进行裁切，最大程度上避免这样的取整对后续操作带来精度上的影响。



当然还有些其他的改动，如在RPN阶段把多尺度特征变成特征金字塔加入了候选的网络中，这个方法采用的是何凯明FPN的论文。还研究了每一个部件，如把RoI Align换成RoI pooling等每一项改进会对精度造成多少影响。

**Mask ^X R-CNN:**

采用迁移学习思路，学习一个映射，试图降低训练数据的获取成本。不是finetune的模式，而是在既有box又有mask这样的数据集上学习这样一个映射，通过box weights和mask weights，学到的是weight transfer function。

