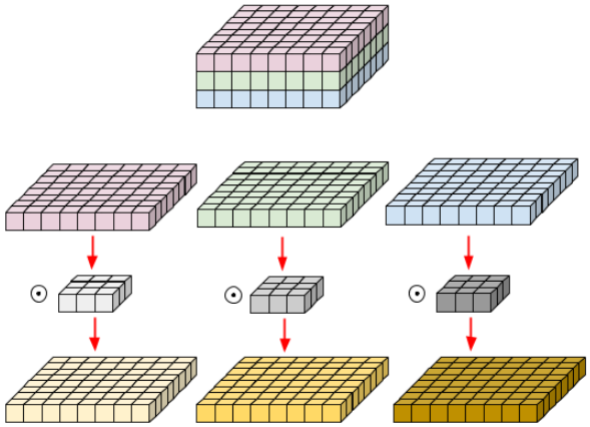
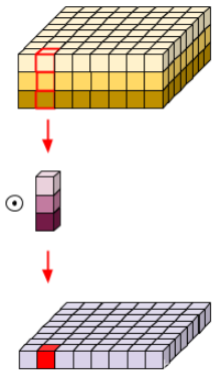
预备知识：深度可分离卷积（MobileNet v1）

深度卷积（Depthwise Conv）：每个卷积核仅对单通道进行卷积，即卷积核大小为 K * K * 1，卷积核数量为 C 个，输出特征图为 P * P * C。输出特征图的通道数 N 通过下面的 N 个 1 * 1 卷积实现。



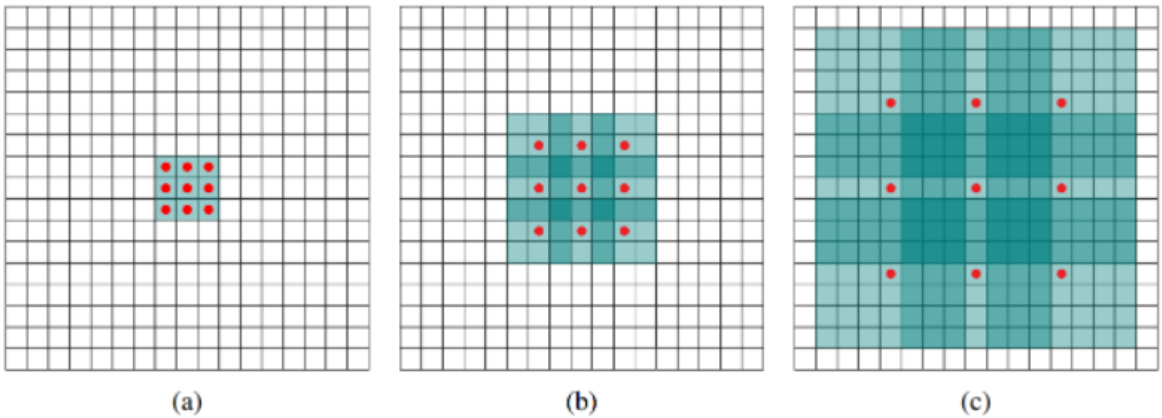
逐点卷积（Pointwise Conv）：用 1 * 1 卷积对深度卷积输出的 C (通道数) 个特征图进行融合，并通过多个该卷积构成特征图的多通道。卷积核大小为 1 * 1 * C，数量为 N 个，输出特征图为 P * P * N。



简单来说将 3 * 3 * C 卷积拆为 3 * 3 * 1 和 1 * 1，不仅模型具有更快的推理速度、更少的参数量，由于层数变深，非线性能力可能更强。

总计算量比值：$[P^2 * C * (N + K^2)] / C * N * K^2 * P^2 = (N + K^2) / (NK^2) = 1/K^2 + 1/N$，若输出特征图通道数 N 越大或卷积核尺寸 K 越大，计算量的比值越小，说明节省的计算量与参数量越多。

空洞卷积：dilated conv



a为普通的3*3卷积，dilation = 1。

b：dilation = 2 的3*3卷积，相当于将感受野扩张为 5 * 5，但除了红点部分外其余权重为0。对标stride为2的3 * 3卷积（有同样大小的输出特征图），有更大的感受野。

c同理，dilation = 4

可以认为 Conv2的参数 dilation 为感受野处相邻两点的坐标差。1为不使用空洞卷积，2即两者间距为1，4即两者间距为3。

增大感受野，在语义分割任务中常用。

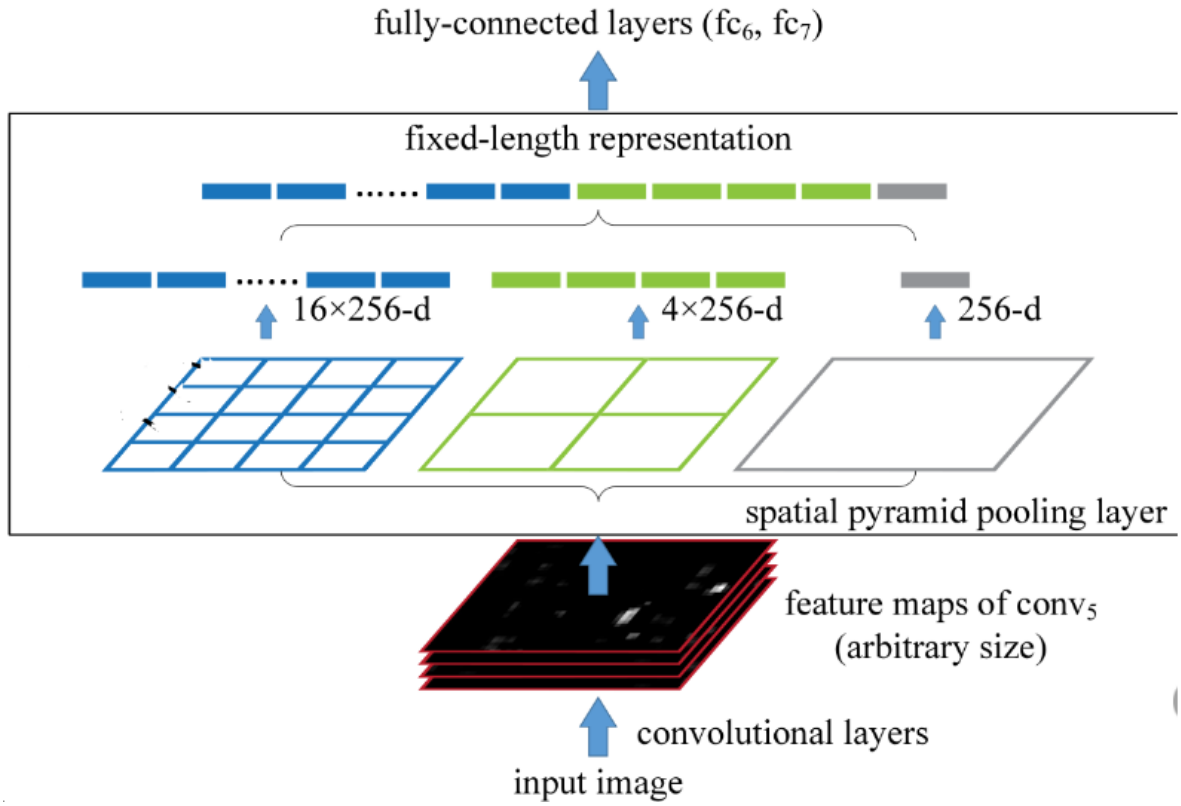
问题：由于卷积核不连续，信息的连续会损失，在 pixel-level 上的任务可能出现问题；且对小物体分割任务而言不利。（解决：HDC）

SPP（Pooling Layer）

SPP：Spatial Pyramid Pooling：空间金字塔池化

基本所有的CNN都要求输入图像的尺寸大小固定，当该尺寸大小不合适时，可能发生：信息丢失/失真（强行resize或切割）、在第一个全连接层 / FCN的输出处维度出错。

SPP 的提出就是为了解决该图像大小必须固定的问题，从而使得图像的长宽比与大小任意，可以适应任何输出，在具有普适性的数据集中非常重要。



此为 SPP 的结构图，SPP 在最后一个卷积层的输出处使用，此时该特征图的尺寸是不定的（但通道数固定）

按照上图，将特征图复制三份，每份作不同尺度的分块，例如蓝色的分为 4 * 4 块，绿色分为 2 * 2 块，灰色分为一整块。对每一个小块作 自适应 Max Pooling（池化的结果为 **H/n * W/n * C -> 1 * 1 * C**），最后将得到 （16 + 4 + 1）* 通道数 的矩阵，即可据此确定全连接层的参数。

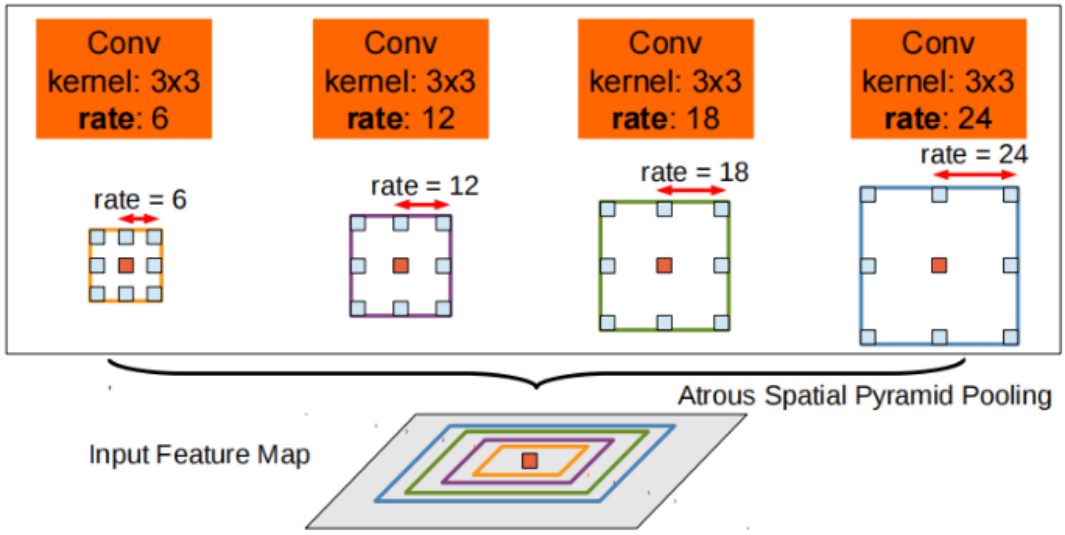
由于同一份特征图中，每个小块的尺寸是一致的，仅需要用一次 MaxPooling，使 卷积核的尺寸与每个小块的尺寸相同，即可完成该操作。

根据上述方法，无论特征图的尺寸或长宽比有何变化，都可以通过分块 + 对整块的池化来控制池化后的维度。这个思想类似 ResNet 中的自适应池化，都是将任意大小的特征图池化为 1 * 1，此处的区别为用类似金字塔的方式，在不同图像块中进行池化。且 ResNet 中的自适应池化通常为均值池化，SPP 通常为最大值池化。

核心计算：池化时卷积核的大小、padding、stride

ASPP：Atrous Spaial Pyramid Pooling

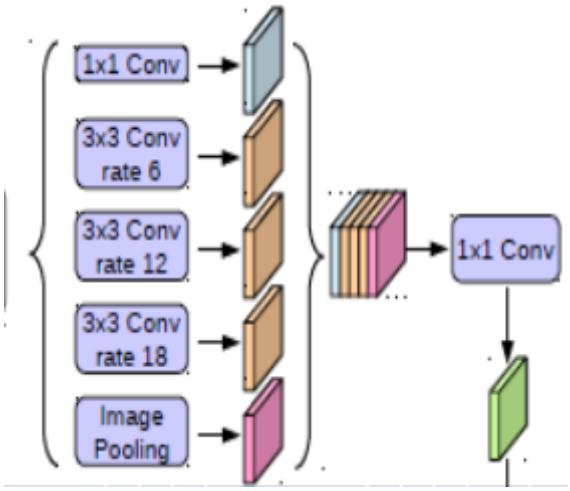
用多个 dilation 不同的空洞卷积核，对图片进行多尺度的池化，同时捕获不同尺度目标的信息。



这里 ASPP 中的池化（降采样）是通过空洞卷积来完成的，并没有真正意义上的降低图片分辨率以获得更大感受野，而是通过空洞卷积获得大感受野后使用 1 * 1 卷积进行通道融合，该模块的输出分辨率与原特征图一致。

ASPP 共有五个尺度：

1. 1 * 1 卷积（只对滤波器中心起作用，尺度不变）；
2. 第二到四个为 3 * 3 空洞卷积（rate = 6,12,18，调整于 DeepLabv3），捕捉局部大尺度信息
3. Image Pooling（即 ResNet 中的 Adaptive Average Pooling）+ 上采样，负责提取全局信息（这里平均池化的输出维度有待考究，在 DeepLabv3 中引入）。



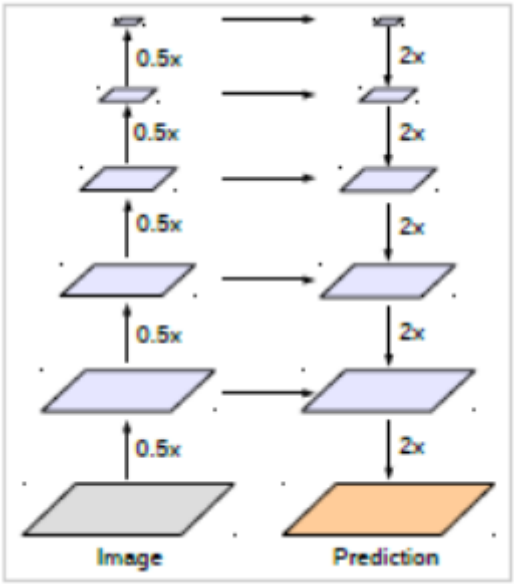
将得到的五张特征图 **cat** 后，使用 1 * 1 卷积融合，得到输出（即 DeepLabv2 ~ DeepLabv3+中，Encoder 的输出）

该文章中提到，在 FCN 提出后，直接各个通道相加的操作减少了，大多数都使用 cat + 1 * 1 卷积进行融合。

另外，在 DeepLabv3 中，ASPP 模块中引入了 BN 层

Encoder - Decoder 结构

CNN 中的 编解码模型通常如下：



可见常用于 gt 与 image 尺寸相同，或其他需要上采样的任务中，例如显著性检测、分割任务等。

其中 Encoder 部分与其他 CNN backbone 结构类似，通过多级下采样增大感受野，提取多尺度信息。

Decoder 部分常通过 Encoder 处输出的特征图经过多级上采样，并与 Encoder 部分对应同尺度（同分辨率）的特征图进行**融合**后得到。

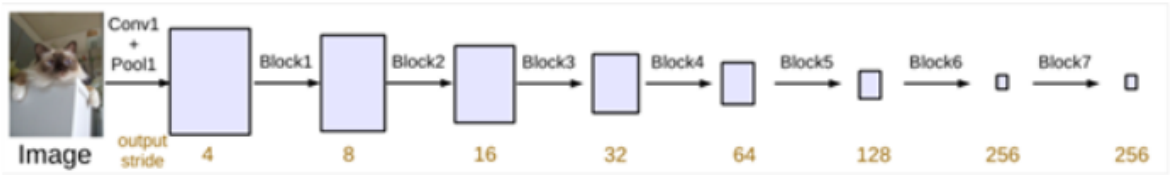
DeepLabV3

encoder-decoder、深度可分离卷积 DeepLabV3+中使用，此处尝试先复现 DeepLabV3，不再考虑。

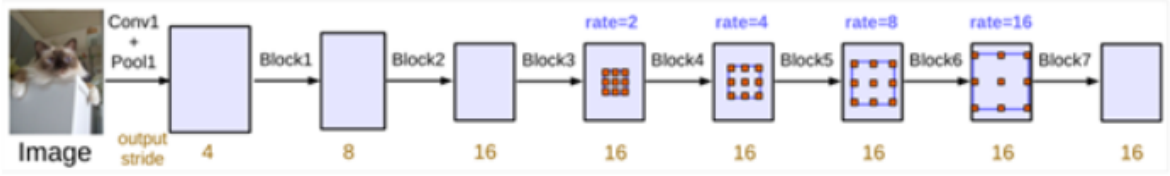
Backbone

主干网为基于 ResNet50 加上空洞卷积后的改版。

1. 其中第一个卷积头中的 7 * 7 卷积改为 ResNetv1c 中的 3 个 3 * 3 卷积叠加。
2. 在步长非 1 的卷积块中（即带下采样的卷积块），使用空洞卷积代替，具有同样的感受野且不损失图像分辨率。



(a) Going deeper without atrous convolution.



(b) Going deeper with atrous convolution. Atrous convolution with *rate* > 1 is applied after block3 when *output_stride* = 16.

Figure 3. Cascaded modules without and with atrous convolution.

此处主要借鉴了博客中提到的一系列改动，具体的细节见代码

Backbone

- 原始的ResNet-50中4个stage的strides=(1, 2, 2, 2)，不采用膨胀卷积即dilations=(1, 1, 1, 1)，而在FCN中4个stage的strides=(1, 2, 1, 1)，dilations=(1, 1, 2, 4)。
- 另外有一个contract_dilation=True的设置，即当空洞>1时，压缩第一个卷积层。这里在第三个和第四个stage的第一个bottleneck中将膨胀率减半，即第三个stage的第一个bottleneck中不采用膨胀卷积，第四个stage的第一个bottleneck中dilation=4/2=2。
- 另外这里采用的是ResNetV1c，即stem中的7x7卷积替换成了3个3x3卷积。
- 最后，注意一下padding，在原始实现中除了stem中7x7卷积的padding=3，其它所有padding=1。在FCN中因为用了膨胀卷积，后两个stage的stride=1，为了保持输入输出分辨率一直，padding=dilation。
- 假设batch_size=4，模型输入shape=(4, 3, 480, 480)，则backbone四个stage的输出分别为(4, 256, 120, 120)、(4, 512, 60, 60)、(4, 1024, 60, 60)、(4, 2048, 60, 60)。

ASPP Head

由上述五个尺度组成，由于训练时出现明显过拟合，在 ASPP 头中添加了概率为 0.1 的 Dropout。

与前面对 ASPP 描述不同的是，博客中在最后整合输出时，不是直接使用 1 * 1卷积控制通道数，而是在 1 * 1 卷积前增加了一个 3 * 3 卷积。

取ResNet第四个stage的输出(4, 2048, 60, 60)作为aspp head的输入。首先经过全局平均池化得到(4, 2048, 1, 1)，然后经过1x1卷积得到(4, 512, 1, 1)，最后通过bilinear插值再上采样回去，得到该分支的输出(4, 512, 60, 60)。

然后1x1分支和3个3x3膨胀卷积分支的输出维度都为(4, 512, 60, 60)，注意这里output_stride=8，相比于上面介绍中的16，这里的膨胀率也要加倍分别为12、24、36。

将5个分支的输出拼接得到(4, 2560, 60, 60)，然后经过3x3卷积得到(4, 512, 60, 60)。

采用dropout，dropout_ratio=0.1。

最后经过1x1卷积得到模型的输出(4, num_classes, 60, 60)，这里num_classes包括背景类。

Output

在 ASPP 头输出后使用双线性插值还原输入。

由于此次任务指定的数据集 Weizmann Horse 中仅存在两类（马与背景），故使用二元交叉熵损失函数即可。（此时任务与 显著性检测 类似）

IOU

交并比的具体计算实现：在目标检测任务（回归框）时，使用矩形相交区域计算即可

而如显著性检测、语义分割任务中，gt 的形状无法估计，使用两个 mask 直接进行计算

gt 与 pred 都为 float16 时，对其进行向量乘（np：gt * pred），即可得到其交集（∩）

对其进行向量加（np：gt + pred），并减去交集（- gt * pred），即可得到并集（∪）

用上述两集合面积（np.sum）相除即可得到 IOU。

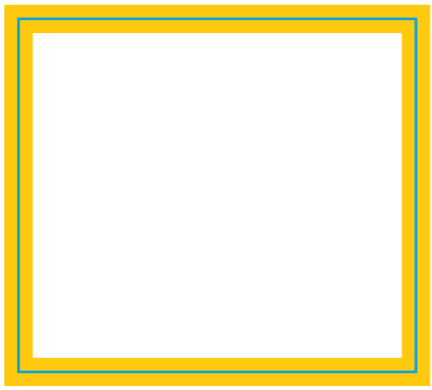
Boundary IOU

相较于 IOU，更关注分割的边界。此处 BIOU 的计算公式为：

$$\frac{|(G_d \cap G) \cap (P_d \cap P)|}{|(G_d \cap G) \cup (P_d \cap P)|}$$

其中 G 为 gt，P 为 pred，G_d 与 P_d 分别表示 gt 与 pred 的边缘。

G_d 与 G 取交集，意思为仅关注 **目标内部的边缘**：



例如上图中 蓝色框内为 gt，左边的黄色框为经膨胀后的整体边缘，右图中为 G_d 与 G 的交集，即前景内部的边缘。

具体实现：此处我的实现基于 opencv，先使用 canny算子检测边缘（此时得到的边缘较薄，可近似视为单像素边缘），再使用 dilate 对边缘进行膨胀，得到上例中左侧的黄框。将边缘与 pred 和 gt 取交集后，即可计算 BIOU。