本周阅读论文《Strip Pooling: Rethinking Spatial Pooling for Scene Parsing》以及《Cars Can't Fly up in the Sky: Improving Urban-Scene Segmentation via Height-driven Attention Networks》论文的相关理解和代码理解。

Strip Pooling: Rethinking Spatial Pooling for Scene Parsing

论文重要的贡献如下:

1、研究了传统的空间池化设计，提出了条形池化的概念，它继承了全局平均池化的优点，既能收集长期依赖关系，又能关注局部细节。

2、设计了基于Strip Pooling的Strip Pooling模块和混合池化模块。这两个模块都是轻量级的，可以作为高效的附加模块插入到任何主干网络中，以生成高质量的分段预测。

3、在SPNet上使用两个基于池化的模块集成到一个单一的体系结构中，在基线上取得了显著的改进，并在广泛使用的场景解析基准数据集上建立了新的最先进的结果。

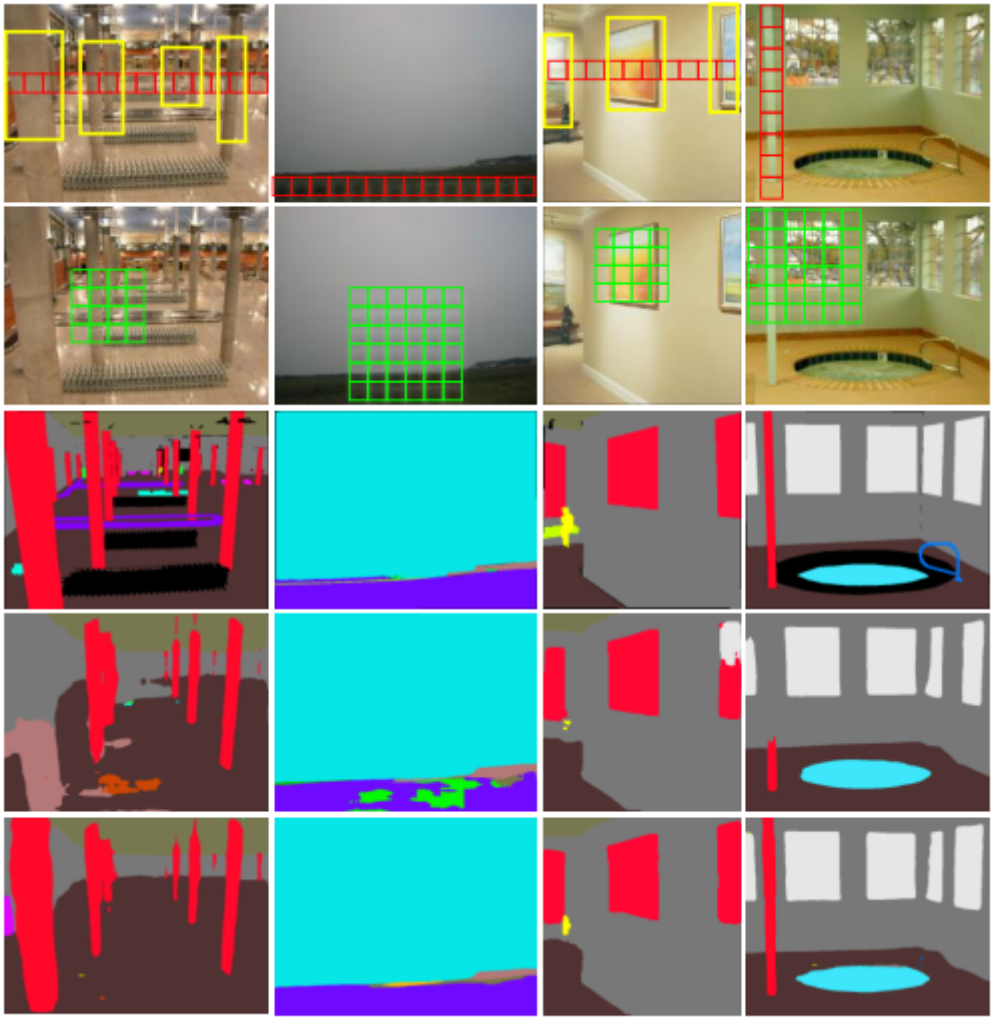
摘要

对于像素级预测任务(如场景分析)而言，空间池在捕获远程上下文信息方面非常有效。作者不再使用传统的$N \times N$规则的空间池化而是使用$1 \times N$或者$N \times 1$的卷积核。进而重新设计了新的空间池化的公式。在带区池化的基础上，进一步研究了空间池化体系结构设计：1)引入了一种新的带区池化模块，使骨干网络能够有效地建模远程依赖关系；2)提出了一种以不同空间池化为核心的新型构建块；3)系统地比较了所提出的带区池化技术与传统空间池化技术的性能。这两种新颖的基于池的设计都是轻量级的，可以作为现有场景解析网络中的一个高效的即插即用模块。在流行基准(例如，ADE20K和CitysScenes)上的广泛实验表明，作者的简单方法建立了新的最先进的结果。

1、介绍

场景分析，也称为语义分割，旨在为图像中的每个像素分配一个语义标签。作为最基本的任务之一，它已被广泛应用于计算机视觉和图形应用，例如自动驾驶、医疗诊断、图像/视频编辑、显著目标检测和航空图像分析。近年来，基于完全卷积网络(FCNs)的方法以其捕捉高级语义的能力，在场景解析方面取得了非凡的进展。然而，这些方法大多将局部卷积和池化操作叠加在一起，因此由于有效视场有限，很难很好地处理各种不同类别的复杂场景。

提高卷机网络的长程依赖建模能力的一种方法是采用自我注意力机制或者non-local modules。但是在计算每个空间位置的时候消耗大量的内存。其他用于远程上下文建模的就返回包括：扩展卷积（在不引入额外参数的情况下扩宽CNN的接受范围）。或者使用全局或者金字塔池化。但是这些方法都是要求使用正方形的窗口映射。这种情况就限制了真实场景中各向异性上下文的灵活性。例如 在图一所示的具体对象具有长范围的带状结构。



为了满足更高效、更有效地捕捉长期依赖关系。作者利用空间池来扩大卷积神经网络的接受域和手机上下文的能力并且提出了条形池的概念。它可以捕获孤立区域的长期关系，如图1a和1c的顶部所示。次，它在另一个空间维度上保持了较窄的核形状，便于捕获局部上下文，并防止不相关区域干扰标签预测。集成这种长而窄的池化内核使得场景解析网络能够同时聚合全局和局部上下文。这与从固定正方形区域收集上下文的传统空间池有本质上的不同。

带状池化操作的基础上，作者提出了两种基于池化的场景解析网络模块。首先，我们设计了一个条形池模块(SPM)来有效地扩大主干网的接受范围。此外，我们还提出了一种新的附加残差构建块，称为混合池模块(MPM)，以进一步在高层语义层次上对长期依赖关系进行建模。它通过利用不同内核形状的汇集操作来探测具有复杂场景的图像，从而收集信息丰富的上下文信息。作者将这两个模块池化到ResNet主干中的SPNET。实验表明，SPNet在流行的场景解析基准上建立了新的最先进的结果。

2、相关工作

当前先进的场景解析（语义分割）都是基于卷及神经网络的，但是由于CNN是通过堆叠局部卷积或者池化算子导致了增长缓慢，无法考虑足够的上下文信息。对场景解析的上下文关系进行建模的早期技术例如条件随机场（CRF系列）。它们大多都是离散的便签中进行建模，计算代价过大。

对于连续的特征空间学习，现有的工作大多使用多尺度特征聚合来融合上下文信息方法是以多速率、多视场探测输入特征或池化操作。DeepLab及其后续采用扩张式卷积，融合不同扩张率特征来增加网络的接受场。

改善感受野的另一项研究是空间金字塔汇集。通过在每个金字塔级别采用一组具有唯一内核大小的并行池操作，网络能够捕获更大范围的上下文。它已经在几个场景解析基准测试中显示出了良好的前景。然而，它利用上下文信息的能力是有限的，因为只应用了正方形的内核形状。此外，空间金字塔池仅在骨干网络之上模块化，因此不灵活或不能直接应用于用于特征学习的网络构建块中。

3、方法论

在本节中首先给出了条带池的概念，然后介绍了两个基于条带池的模型设计，以演示它是如何改进场景解析网络的,最后，描述了该场景解析网络的整体架构，并对其进行了条带池的扩展。

3.1 Strip pooling

本届主要介绍了如何进行Strip Pooling的操作。

标准的平均空间池化： 设$x \in \mathbb{R}^{H \times W}$ 为一个二维输入张量，其中H和W分别为空间的高度和宽度。在平均池化层中需要将汇集的$h \times w$ 考虑一个简单的情况，其中h除以H，w除以W。

形式上平均池化可以写成：

$$y_{i_o,j_o} = \frac{1}{h \times w} \sum_{0 \le i < h} \sum_{0 \le j < w} x_{i_o \times h + i, j_o \times w + j}, \qquad (1)$$

但是在处理不规则的图像时会存在包含许多不必要区域的问题。

Strip Pooling:了缓解上述问题，我们在这里提出了“Strip Pooling”的概念，它使用带状池化窗口沿水平或垂直维度执行池化。在数学上给定二维向量$x \in \mathbb{R}^{H \times W}$ 在Strip Pooling中需要池化核为(H,1)或者(1,H).与平均池化不同的是所提出的Strip Pooling 会池化一行或者一列的数据。

给定水平和垂直条带池化层，由于长而窄的核形状，很容易在离散分布的区域之间建立远程依赖关系，并利用带状形状对区域进行编码。同时，由于其沿另一个维度的核形状较窄，因此它也侧重于捕捉局部细节。这些性质使得所提出的带状合并不同于传统的依赖于正方形核的空间合并。如下图所示

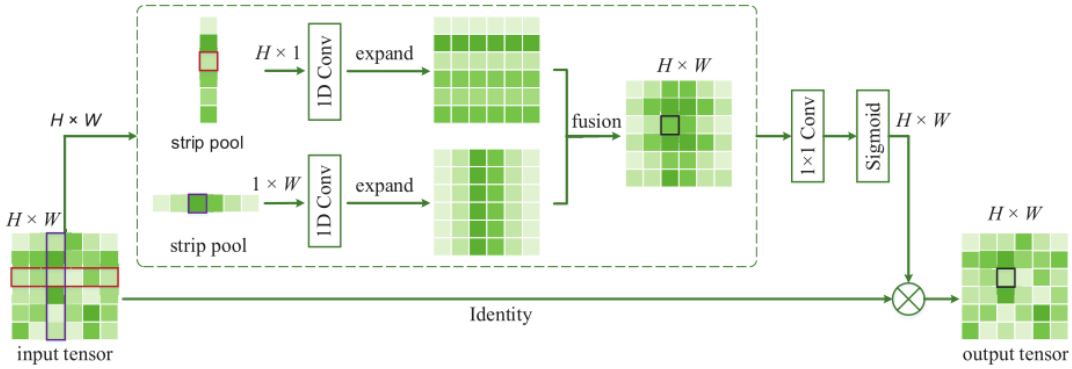


Figure 2. Schematic illustration of the Strip Pooling (SP) module.

3.2 Strip Pooling Module

本节介绍了一种通过利用条带池来帮助主干网络捕获远程上下文的有效方法。特别地，提出了 Strip Pooling Module，它同时利用水平和垂直条带池化操作来收集来自不同空间维度的远程上下文信息。如上图所示。

SPM允许输出张量中的每个位置与输入张量中的各种位置建立关系。例如，在图2中，以输出张量中的黑框为边界的正方形连接到具有与其相同的水平或垂直坐标的所有位置(由红色和紫色方框包围)。因此，通过重复上述聚合过程几次，可以在整个场景中构建远程依赖关系。此外，得益于元素乘法运算，所提出的SPM也可以被认为是一种注意力机制，可以直接应用于任何预先训练的骨干网络，而不需要从头开始训练它们。

与全局平均汇集相比，Strip Pooling考虑了较长但较窄的范围，而不是整个要素地图，避免了在彼此相距较远的位置之间建立大多数不必要的连接。与需要大量计算来建立每对位置之间的关系的基于注意力的模块相比，SPM是轻量级的，可以很容易地嵌入到任何构建块中，以提高捕获远程空间依赖和利用通道间依赖的能力。

3.3 Mixed Pooling Module

考虑到标准空间池化和条带池化的优点，我们提出了PPM，并设计了一种混合池化模块(MPM)，该模块通过各种池化操作来聚合不同类型的上下文信息，使特征表示更具区分性。

所提出的MPM由同时捕获不同位置之间的短距离和长距离依赖性的两个子模块组成.对于长期依赖关系,作者不再像之前的工作那样使用全军平均池化而是采用 Strip Pooling ，简图见图3(B)。如第3.2节中所分析的，条带池使得分散分布在整个场景上的区域之间的连接和使用带状结构的编码区域成为可能。然而，对于语义区域分布紧密的情况，为了捕获局部上下文信息，空间池也是必要的。考虑到这一点，如图3(A)所示，作者采用了一个轻量级金字塔池子模块来进行短距离依赖关系收集。它有两个空间池层，然后是用于多尺度特征提取的卷积层，以及用于保存原始空间信息的2D卷积层。每次汇集后的特征图分别具有20×20和12×12的大小。然后通过求和将所有三个子路径组合在一起。

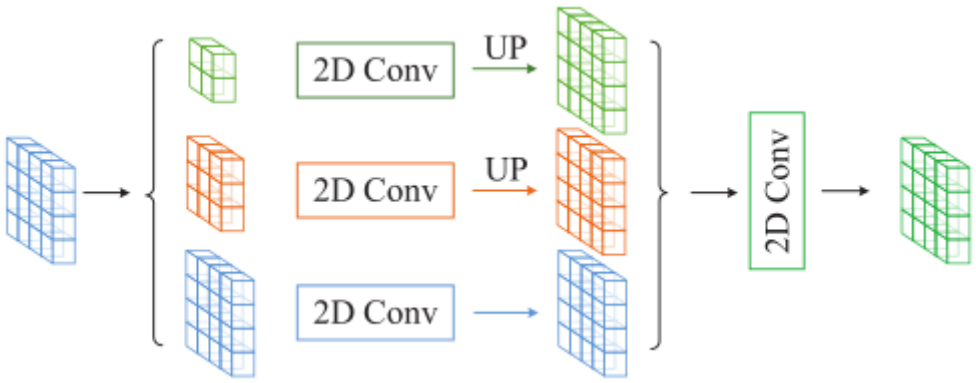
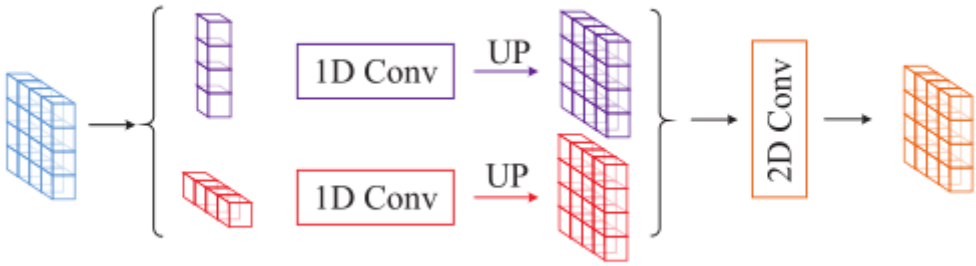


图3

(a)



(b)

3.4 整体架构

在题图SPM和MPM的基础上 提出了SPNet的总体架构，采用经典的的残差网络作为主干并且修改了Restnet 并将最终的特征图大小设置为输入图像的1/8。将SPM添加到每级最后一个构建块的3×3卷积层和最后一级所有构建块的3×3卷积层之后。SPM中的所有卷积层共享到输入张量的相同数目的通道。对于MPM，直接在主干网络上搭建，因为它是模块化的设计。由于骨干网的输出为2048通道，我们先在骨干网上连接1×1卷积层，将2048路减少到1024路，然后再增加两个MPM。在每个MPM中，之后，核大小为3×3或3的所有卷积层具有256个信道(即，使用1/4的缩减率)。最后添加卷积层对分割图进行预测。

4、实验

我们在ADE20K、Cityspaces 和Pascal Context数据集上验证了SPM和MPM有效性。此外还在ADE20K数据集上进行了Strip pooling的全面分析。

4.1 实验设置

网络基于开个开源工具和Pytorch实现，我们使用4GPu进行训练。将Cityspace数据集设置batch size为8 其他的数据集设置batch size为16.与大多数训练一样 我们也采用‘ploy’学习率。将ADE20K的学习率设置为0.004 ,Pascal Context 数据集设置为0.0001 动量设置为0.9 训练的epoch分别为：ADE20K (120), Cityscapes (180), and Pascal Context (100). 将动量和衰减率分贝设置为0.9和0.0001 我们使用synchronized Batch Normlization进行正则化。将输入图像进行0.5的随机翻转和缩放，最后将图像进行重新固定 （Cityspace为784x784 其他为484x484）使用cross-entropy进行优化

Settings	#Params	SPM	mIoU	Pixel Acc
Base FCN	27.7 M	✗	37.63	77.60%
Base FCN + PPM [65]	+21.0 M	✗	41.68	80.04%
Base FCN + 1 MPM	+4.4 M	✗	40.50	79.60%
Base FCN + 2 MPM	+8.8 M	✗	41.92	80.03%
Base FCN + 2 MPM	+11.9 M	✓	**44.03**	**80.65%**

Table 1. Ablation analysis on the number of mixed pooling modules (MPMs). 'SPM' refers to the strip pooling module. As can be seen, when more MPMs are used, better results are yielded. All results are based on ResNet-50 backbone and single-model test. Best result is highlighted in **bold**.

4.2.1 Ablation Studies

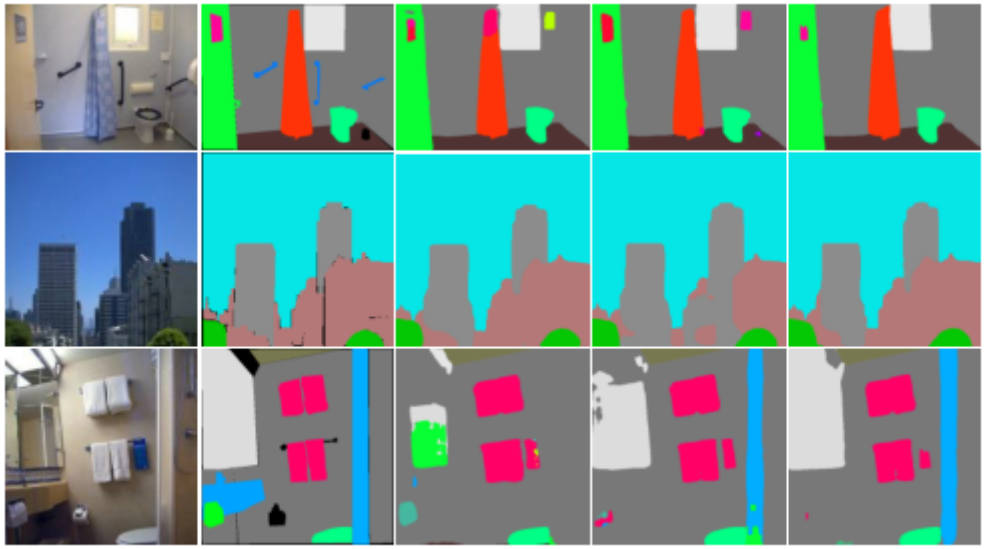
MPM的数量：如第3.3节所述，MPM是基于剩余块的瓶颈结构构建的，因此可以方便地重复多次，以扩展Strip Pooling的作用。测试需要多少MPM来平衡所提出方法的性能和运行时成本。如表1所示，我们列出了基于ResNet-50主干使用不同数量的MPM时的结果。可以看到，当不使用MPM(基本FCN)时，我们在mIoU方面达到了37.63%的结果。当使用1个MPM时，我们的结果是40.50%，即大约3.0%的改进。在增加两个MMP时可以获得大约4.3%的性能提升，但是如果再增加更多的话提升就不明显了。因此我们将MPM数量设置为2.

同时为了希腊式锁提出的MPM相对于PPM的优势，我们还在表1中给出了PSPNet的结果和参数，尽管参数比PSPNet少了12M但是性能确更好。

Settings	w/ SPM	mIoU	Pixel Acc
Base FCN	✗	37.63	77.60%
Base FCN + 2 MPM (SRD only)	✗	40.50	79.34%
Base FCN + 2 MPM (LRD only)	✗	41.14	79.64%
Base FCN + 2 MPM (SRD + LRD)	✗	41.92	80.03%
Base FCN + 2 MPM (SRD + LRD)	✓	44.03	80.65%

Table 2. Ablation analysis on the mixed pooling module (MPM). 'SPM' refers to the strip pooling module. 'SRD' and 'LRD' denote the short-range dependency aggregation sub-module and the long-range dependency aggregation sub-module, respectively. As can be seen, collecting both short-range and long-range dependencies are essential for yielding better segmentation results. All results are based on single-model test.

如上表所示，使用2个MPM并且一个用于长程依赖一个用于短程依赖得到了最好的效果。并将效果进行可视化对比如下图：



(a) Image (b) GT (c) 2 SRD (d) 2 LRD (e) 2 MPM

如果将Strip Pooling改为全局平局池化，在使用Stip Pooling的时候我们mIoU从41.92%增加到了44.3% 但是换成全局平均池化就会从基准的41.92%下降到41.34% 因为全局平均池化丢失了许多细节信息。

4.2.2 与当前先进的结果进行比较

Method	Backbone	mIoU (%)	Pixel Acc. (%)	Score
RefineNet [32]	ResNet-152	40.70	-	-
PSPNet [65]	ResNet-101	43.29	81.39	62.34
PSPNet [65]	ResNet-269	44.94	81.69	63.32
SAC [63]	ResNet-101	44.30	81.86	63.08
EncNet [60]	ResNet-101	44.65	81.69	63.17
DSSPN [30]	ResNet-101	43.68	81.13	62.41
UperNet [52]	ResNet-101	42.66	81.01	61.84
PSANet [66]	ResNet-101	43.77	81.51	62.64
CCNet [23]	ResNet-101	45.22	-	-
APNB [69]	ResNet-101	45.24	-	-
APCNet [19]	ResNet-101	45.38	-	-
SPNet (Ours)	ResNet-50	45.03	81.32	63.18
SPNet (Ours)	ResNet-101	**45.60**	**82.09**	**63.85**

Table 5. Comparisons with the state-of-the-arts on the validation set of ADE20K [68]. We report both mIoU and Pixel Acc. on this benchmark. Best results are highlighted in **bold**.

从上图中我们可以看出以ResNet-50为主干的方法达到了45.03%的MIU值和81.32%的像素准确率，已经优于以往的大多数方法。当我们以ResNet-101为主干时，mIoU和像素精度方面都取得了新的最先进的结果。

Cars Can't Fly up in the Sky: Improving Urban-Scene Segmentation via Height-driven Attention Networks 代码

根据上周的论文实现论文中的思想。论文中主要提到的是高度驱动的注意力网络。HANet根据其高度相关信息为每个单独的行生成每个通道的比例因子$X_l \in \mathbb{R}^{C_l \times H_l \times W_e}$ 和 $X_h \in \mathbb{R}^{C_h \times H_h \times W_h}$ 在语义分割网络中代表低级和高级的特征图。

其网络结构如下所示：

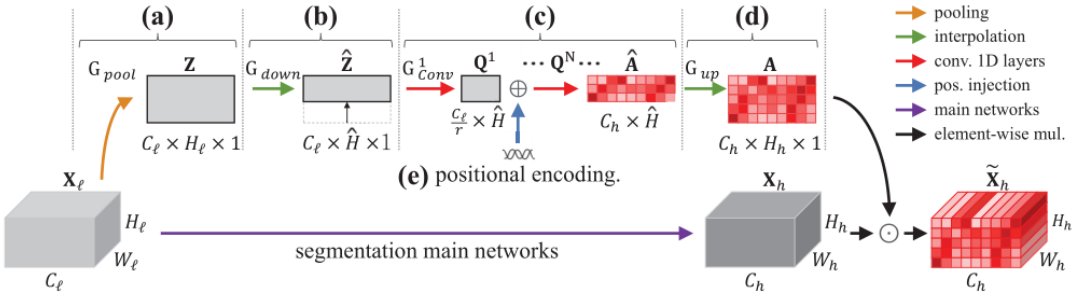


Figure 2: Architecture of our proposed HANet. Each operation *op* is notated as G_{op}, and feature maps are in bold–X_ℓ: lower-level feature map, Z: width-wise pooled X_ℓ, $\hat{Z}$: down-sampled Z, Q^n: n-th intermediate feature map of 1D convolution layers, $\hat{A}$: down-sampled attention map, A: final attention map, X_h: higher-level feature map, $\tilde{X}_h$: transformed new feature map. Details can be found in Section 3.1.

对应的pytorch核心代码如下：

```
    def __init__(self, in_channel, out_channel, kernel_size=3,
r_factor=64, layer=3, pos_injection=2, is_encoding=1,
                 pos_rfactor=8, pooling='mean', dropout_prob=0.0,
pos_noise=0.0):
        super(HANet_Conv, self).__init__()

        self.pooling = pooling
        self.pos_injection = pos_injection
        self.layer = layer
        self.dropout_prob = dropout_prob
        self.sigmoid = nn.Sigmoid()

        if r_factor > 0:
            mid_1_channel = math.ceil(in_channel / r_factor)
        elif r_factor < 0:
            r_factor = r_factor * -1
            mid_1_channel = in_channel * r_factor

        if self.dropout_prob > 0:
            self.dropout = nn.Dropout2d(self.dropout_prob)

        self.attention_first = nn.Sequential(
                nn.Conv1d(in_channels=in_channel,
out_channels=mid_1_channel,
                    kernel_size=1, stride=1, padding=0,
bias=False),
```

```
                Norm2d(mid_1_channel),
                nn.ReLU(inplace=True))

        if layer == 2:
            self.attention_second = nn.Sequential(
                    nn.Conv1d(in_channels=mid_1_channel,
out_channels=out_channel,
                        kernel_size=kernel_size, stride=1,
padding=kernel_size//2, bias=True))
        elif layer == 3:
            mid_2_channel = (mid_1_channel * 2)
            self.attention_second = nn.Sequential(
                    nn.Conv1d(in_channels=mid_1_channel,
out_channels=mid_2_channel,
                        kernel_size=3, stride=1, padding=1,
bias=True),
                    Norm2d(mid_2_channel),
                    nn.ReLU(inplace=True))
            self.attention_third = nn.Sequential(
                    nn.Conv1d(in_channels=mid_2_channel,
out_channels=out_channel,
                        kernel_size=kernel_size, stride=1,
padding=kernel_size//2, bias=True))

        if self.pooling == 'mean':
            #print("##### average pooling")
            self.rowpool =
nn.AdaptiveAvgPool2d((128//pos_rfactor,1))
        else:
            #print("##### max pooling")
            self.rowpool =
nn.AdaptiveMaxPool2d((128//pos_rfactor,1))

        if pos_rfactor > 0:
            if is_encoding == 0:
                if self.pos_injection == 1:
                    self.pos_emb1d_1st =
PosEmbedding1D(pos_rfactor, dim=in_channel, pos_noise=pos_noise)
                elif self.pos_injection == 2:
                    self.pos_emb1d_2nd =
PosEmbedding1D(pos_rfactor, dim=mid_1_channel, pos_noise=pos_noise)
            elif is_encoding == 1:
                if self.pos_injection == 1:
                    self.pos_emb1d_1st = PosEncoding1D(pos_rfactor,
dim=in_channel, pos_noise=pos_noise)
                elif self.pos_injection == 2:
                    self.pos_emb1d_2nd = PosEncoding1D(pos_rfactor,
dim=mid_1_channel, pos_noise=pos_noise)
            else:
                print("Not supported position encoding")
                exit
```

实现的损失函数如下：

```
```

```
    if args.img_wt_loss:
        criterion = ImageBasedCrossEntropyLoss2d(
            classes=args.dataset_cls.num_classes, size_average=True,
            ignore_index=args.dataset_cls.ignore_label,
            upper_bound=args.wt_bound).cuda()
    elif args.jointwtborder:
        criterion = ImgWtLossSoftNLL_by_epoch(classes=args.dataset_cls.num_classes,
ignore_index=args.dataset_cls.ignore_label,
upper_bound=args.wt_bound).cuda()
    else:
        criterion = CrossEntropyLoss2d(size_average=True,
ignore_index=args.dataset_cls.ignore_label).cuda()

    criterion_val = CrossEntropyLoss2d(size_average=True,
                                       weight=None,
ignore_index=args.dataset_cls.ignore_label).cuda()
    return criterion, criterion_val
```

可以将HANet理解为一个插件，可以插入在原有的网络上进一步提高之前网络的性能与准确性。论文中以DeepV3为例子，其实现代码如下：

```
class DeepV3PlusHANet(nn.Module):
    """
    Implement DeepLab-V3 model
    A: stride8
    B: stride16
    with skip connections
    """

    def __init__(self, num_classes, trunk='resnet-101', criterion=None, criterion_aux=None,
                 variant='D', skip='m1', skip_num=48, args=None):
        super(DeepV3PlusHANet, self).__init__()
        self.criterion = criterion
        self.criterion_aux = criterion_aux
        self.variant = variant
        self.args = args
        self.num_attention_layer = 0
        self.trunk = trunk
        # 省略部分源码
            self.layer0 = resnet.layer0
            self.layer1, self.layer2, self.layer3, self.layer4 = \
                resnet.layer1, resnet.layer2, resnet.layer3, resnet.layer4

            if self.variant == 'D':
                for n, m in self.layer3.named_modules():
                    if 'conv2' in n:
```

```
                        m.dilation, m.padding, m.stride = (2, 2),
(2, 2), (1, 1)
                    elif 'downsample.0' in n:
                        m.stride = (1, 1)
                for n, m in self.layer4.named_modules():
                    if 'conv2' in n:
                        m.dilation, m.padding, m.stride = (4, 4),
(4, 4), (1, 1)
                    elif 'downsample.0' in n:
                        m.stride = (1, 1)
            elif self.variant == 'D4':
                for n, m in self.layer2.named_modules():
                    if 'conv2' in n:
                        m.dilation, m.padding, m.stride = (2, 2),
(2, 2), (1, 1)
                    elif 'downsample.0' in n:
                        m.stride = (1, 1)
                for n, m in self.layer3.named_modules():
                    if 'conv2' in n:
                        m.dilation, m.padding, m.stride = (4, 4),
(4, 4), (1, 1)
                    elif 'downsample.0' in n:
                        m.stride = (1, 1)
                for n, m in self.layer4.named_modules():
                    if 'conv2' in n:
                        m.dilation, m.padding, m.stride = (8, 8),
(8, 8), (1, 1)
                    elif 'downsample.0' in n:
                        m.stride = (1, 1)
            elif self.variant == 'D16':
                for n, m in self.layer4.named_modules():
                    if 'conv2' in n:
                        m.dilation, m.padding, m.stride = (2, 2),
(2, 2), (1, 1)
                    elif 'downsample.0' in n:
                        m.stride = (1, 1)
            else:
                # raise 'unknown deepv3 variant:
{}'.format(self.variant)
                print("Not using Dilation ")

        if self.variant == 'D':
            os = 8
        elif self.variant == 'D4':
            os = 4
        elif self.variant == 'D16':
            os = 16
        else:
            os = 32
```