CA-Net

论文的研究内容

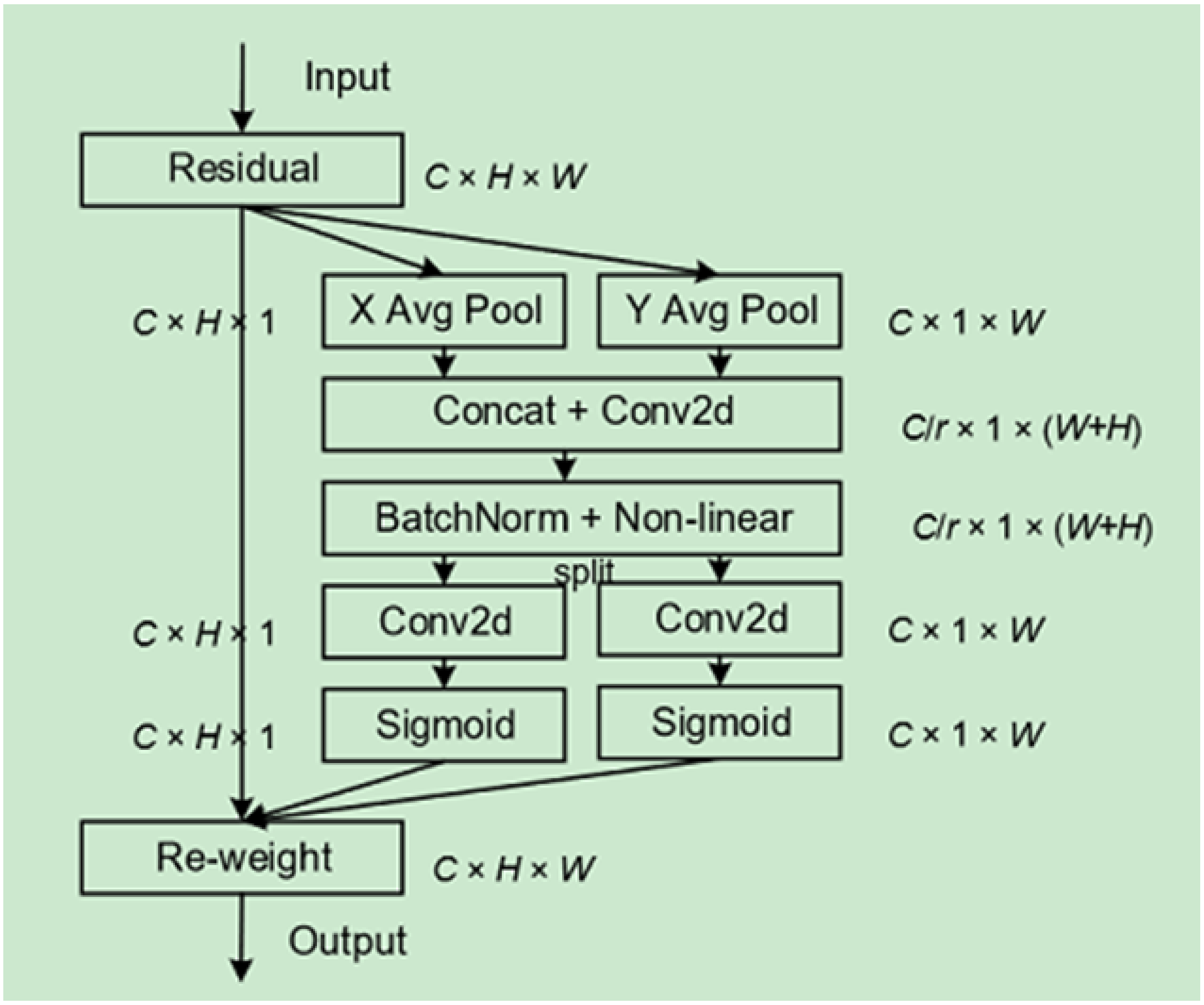
论文原文：

“In this paper, we propose a novel attention mechanism for mobile networks by embedding positional information into channel attention, which we call “coordinate attention”. Unlike channel attention that transforms a feature tensor to a single feature vector via 2D global pooling, the coordinate attention factorizes channel attention into two 1D feature encoding processes that aggregate features along the two spatial directions, respectively. In this way, long-range dependencies can be captured along one spatial direction and meanwhile precise positional information can be preserved along the other spatial direction. The resulting feature maps are then encoded separately into a pair of direction-aware and position-sensitive attention maps that can be complementarily applied to the input feature map to augment the representations of the objects of interest.”

总结：

作者提出一种坐标注意力机制：在h、w空间方向分别进行1维卷积，得到h和w空间方向的注意力图。

原理



1. CA首先沿h、w方向进行avg_pool，将特征图分割成2个特征子图，h方向的特征图维度为b×c×1×w，w方向的特征图维度为b×c×h×1。
2. 将h和w方向特征图沿w方向进行拼接（有一个方向的特征图要进行维度交换，见代码）和通道数压缩
这里有一个想法，后续实现：
结合ECA论文（link: ECA-Net）的思路，可以去掉通道数压缩
3. 特征图经过BN层（归一化）和ReLU
4. 将拼接的特征图进行分割，并利用二维卷积还原通道数，再经过Sigmoid，增加非线性

pytorch代码

PYTHON

```
class CA(nn.Module):
    def __init__(self, c1, c2, ratio=3):
        super(CA, self).__init__()
        self.h_avg_pool = nn.AdaptiveAvgPool2d((1,
None))
        self.w_avg_pool =
nn.AdaptiveAvgPool2d((None, 1))
        self.conv_11 = nn.Conv2d(c1, c1//ratio,
kernel_size=1, stride=1, bias=False)
        self.f_h = nn.Conv2d(c1//ratio, c1,
kernel_size=1, stride=1, bias=False)
        self.f_w = nn.Conv2d(c1//ratio, c1,
kernel_size=1, stride=1, bias=False)

        self.bn = nn.BatchNorm2d(c1//ratio)
        self.relu = nn.ReLU()
        self.sigmoid_h = nn.Sigmoid()
        self.sigmoid_w = nn.Sigmoid()

    def forward(self, x):
        b, c, h, w = x.size()

        # h方向avg_pool: b*c*1*w
        h_avg = self.h_avg_pool(x)

        # w方向avg_pool: b*c*h*1
        w_avg = self.w_avg_pool(x)
        # 维度交换: b*c*h*1->b*c*1*h
        w_avg = w_avg.permute(0, 1, 3, 2)

        # 拼接 & 通道压缩
        avg_out = torch.cat([h_avg, w_avg], dim=3)
        avg_out = self.conv_11(avg_out)
        # BN & relu
        avg_out = self.relu(self.bn(avg_out))

        # split
        h_split, w_split = torch.split(avg_out, [w,
h], dim=3)

        # 维度交换: b*c*1*h->b*c*h*1
        w_split = w_split.permute(0, 1, 3, 2)

        # 通道数还原 & sigmoid
        w_out = self.sigmoid_w(self.f_w(w_split))
        h_out = self.sigmoid_h(self.f_h(h_split))

        return x * h_out * w_out
```