

SEBlock

论文地址：<https://arxiv.org/abs/1709.01507>

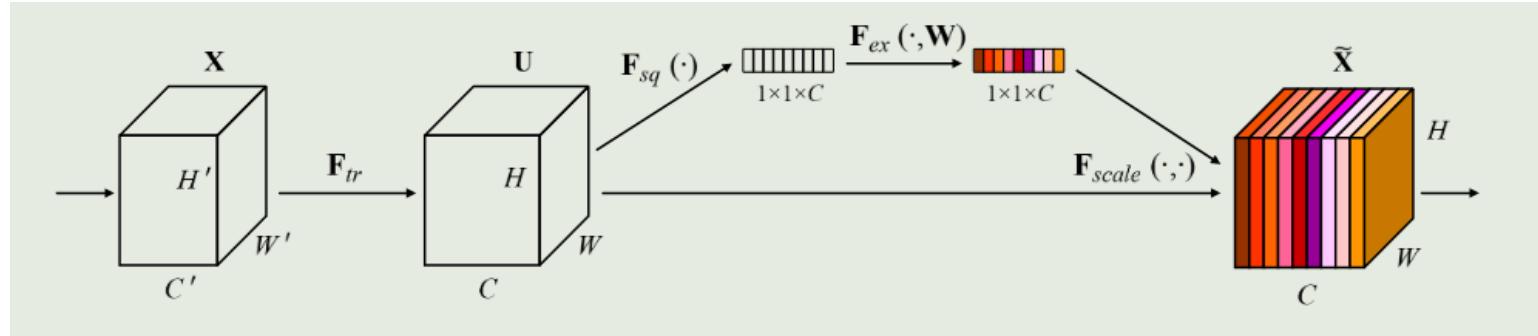
1. 简介

SE注意力模块是一种通道注意力模块，SE模块能对输入特征图进行通道特征加强，且不改变输入特征图的大小。

- SE模块的S(Squeeze):对输入特征图的空间信息进行压缩
- SE模块的E(Excitation):学习到的通道注意力信息，与输入特征图进行结合，最终得到具有通道注意力的特征图
- SE模块的作用是在保留原始特征的基础上，通过学习不同通道之间的关系，提高模型的表现能力。在卷积神经网络中，通过引入SE模块，可以动态地调整不同通道的权重，从而提高模型的表现能力。

直白的说，SE就是对特征图的各个通道施加不同的权重，自动实现重要通道的选择。

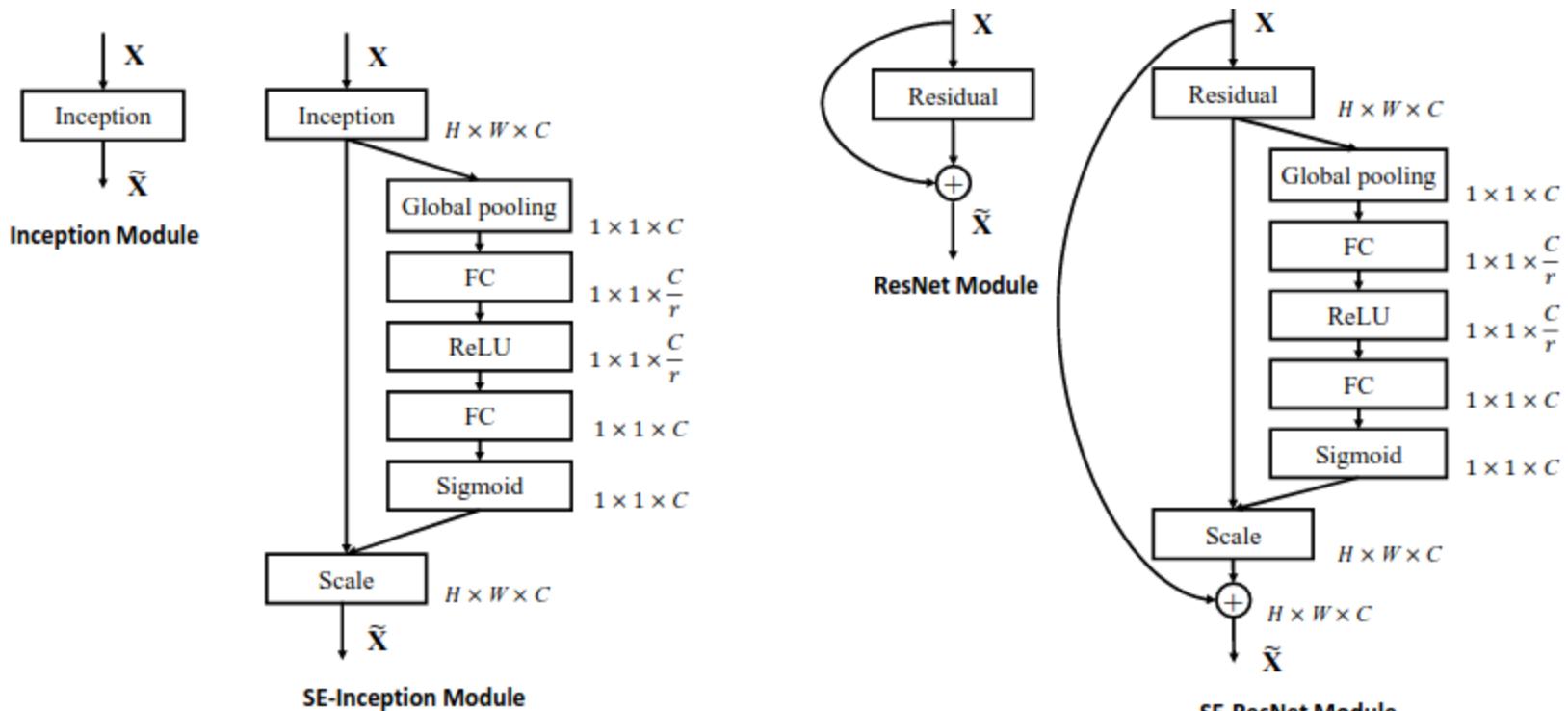
2. 结构



在还没进行SE模块之前，特征图的维度为 $H \times W \times C$

1. 先对特征图进行空间特征压缩，在空间维度上实现全局平均池化，得到 $1 \times 1 \times C$ 的特征图，也就是全局平均池化将每一个通道上的特征图全部压缩为一个数值，该数值具有全局感受野。
2. 首先通过 FC 全连接层学习，将通道数 C 降低至 $C * \text{ration}$ ，这个 ration 是通道数的缩放系数，减少通道个数从而降低计算量。然后将 $C * \text{ration}$ 通道数升高至 C ，实际上就是一个降维再升维的作用。得到最终的注意力特征图，他的维度还是 $1 \times 1 \times C$ 。
3. 将通道注意力 $1 \times 1 \times C$ 的特征图、原始输入 $H \times W \times C$ 的特征图，进行逐通道乘以权重系数，最终输出具有通道注意力的特征图。

降维和升维的操作可以使用1*1的卷积来实现，并且参数量很小。下面右图是将SE模块应用到ResNet网络中去。



具如何起作用的？

权重 s 会与原特征逐通道相乘：

$$\tilde{x}_c = s_c \cdot x_c$$

然后流入后续网络、计算 loss，例如分类 loss (CE)、人脸识别 loss (ArcFace/CosFace/PartialFC)。

在反向传播时：

- 如果某个通道 **有助于降低 loss**，梯度会推动 s_c 变大（更接近 1）。
- 如果某个通道 **对任务无用甚至有害**，梯度会推动 s_c 变小（更接近 0）。

这就是“注意力 (Attention)”的本质。SE 通过 loss 的反向传播，自然就会学会哪些通道应该放大，哪些通道应该抑制。

代码实现：

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class SEBlock(nn.Module):
    """
    Reference:https://arxiv.org/abs/1709.01507
    就是简单的降维之后再升维，计算出个通道对应的权重，返回输入和权重的乘积，也就是加权
    """

    def __init__(self, input_channel: int, ration: float):
        super().__init__()
        self.in_channel = input_channel
        self.scale_ration = ration
        self.reduce = nn.Conv2d( # 1*1降维卷积
            in_channels=self.in_channel,
            out_channels=int(self.in_channel * self.scale_ration), # 输出通道数是对输入通道数缩放ration倍数
            kernel_size=1,
            stride=1,
            bias=False
        )
        self.expand = nn.Conv2d( # 1*1升维卷积
            in_channels=int(self.in_channel * self.scale_ration), # 降维之后的通道数作为输入
            out_channels=self.in_channel,
            kernel_size=1,
            stride=1,
            bias=False
        )

    def forward(self, inputs: torch.Tensor):
        b, c, _, _ = inputs.size()
        x = F.adaptive_avg_pool2d(inputs, 1) # 池化之后每个通道对应一个数值
        x = self.reduce(x)
        x = F.relu(x)
        x = self.expand(x)
        x = torch.sigmoid(x)
        x = x.view(b, c, 1, 1) # 得到的x是权重向量
        return inputs * x
```