训练技巧与模块小结

SR方法仓库 [repo]

即插即用的Attention模块小结 [repo]

14种卷积及变体 [repo]

Awesome超分方法 [repo]

即插即用的CNN小插件 [article]

模块

1-LFB模块

[paper] [code]

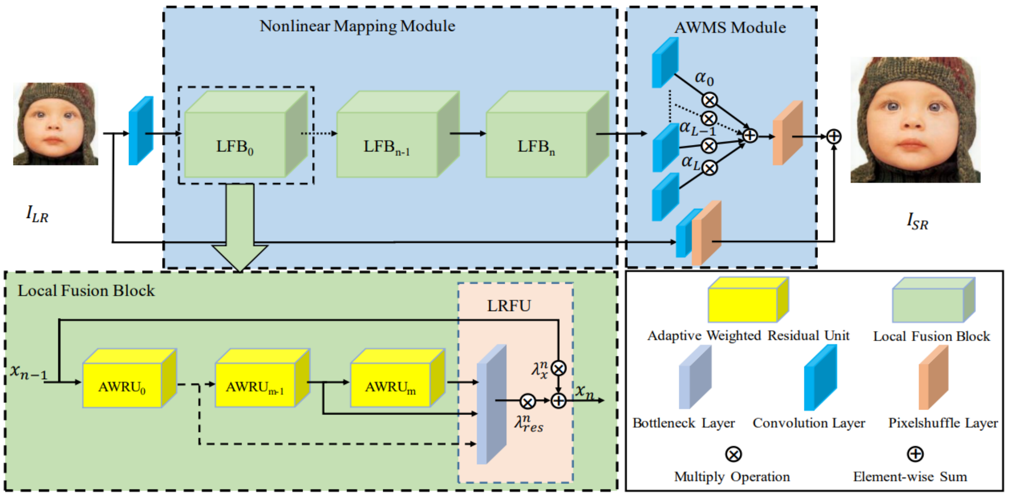


Figure 2. Network architecture of the proposed AWSRN

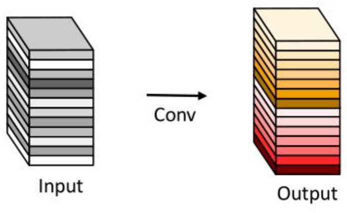
特点：LFB包括：多个堆叠的AWRU和1个LRFU

AWRU基于残差单元，增加了两个独立的权重，赋初值后可以自适应地学习得到。LRFU用于融合AWRU提取到的特征信息。

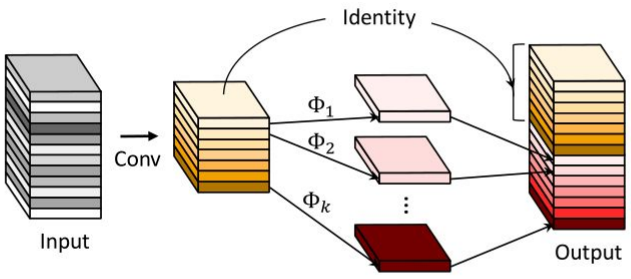
```
wn = lambda x: torch.nn.utils.weight_norm(x)
class Scale(nn.Module):

    def __init__(self, init_value=1e-3):
        super().__init__()
        self.scale = nn.Parameter(torch.FloatTensor([init_value]))

    def forward(self, input):
        return input * self.scale


class AWRU(nn.Module):
    def __init__(
        self, n_feats, kernel_size, block_feats, wn, res_scale=1,
act=nn.ReLU(True)):
        super(AWRU, self).__init__()
        self.res_scale = Scale(1)
        self.x_scale = Scale(1)
        body = []
        body.append(
            wn(nn.Conv2d(n_feats, block_feats, kernel_size,
padding=kernel_size//2)))
        body.append(act)
        body.append(
            wn(nn.Conv2d(block_feats, n_feats, kernel_size,
padding=kernel_size//2)))

        self.body = nn.Sequential(*body)

    def forward(self, x):
        res = self.res_scale(self.body(x)) + self.x_scale(x)
        return res


class LFB(nn.Module):
    def __init__(
        self, n_feats, kernel_size, block_feats, wn, act=nn.ReLU(True)):
        super(LFB, self).__init__()
        self.b0 = AWRU(n_feats, kernel_size, block_feats, wn=wn, act=act)
        self.b1 = AWRU(n_feats, kernel_size, block_feats, wn=wn, act=act)
        self.b2 = AWRU(n_feats, kernel_size, block_feats, wn=wn, act=act)
        self.b3 = AWRU(n_feats, kernel_size, block_feats, wn=wn, act=act)
        self.reduction = wn(nn.Conv2d(n_feats*4, n_feats, 3, padding=3//2))
        self.res_scale = Scale(1)
        self.x_scale = Scale(1)

    def forward(self, x):
        x0 = self.b0(x)
        x1 = self.b1(x0)
        x2 = self.b2(x1)
        x3 = self.b3(x2)
        res = self.reduction(torch.cat([x0, x1, x2, x3],dim=1))

        return self.res_scale(res) + self.x_scale(x)
```

2-ghost module

[paper] [code]



(a) The convolutional layer.



(b) The Ghost module.

特点：利用“廉价的线性运算”（类似深度可分离卷积）生成“幻影”特征图，节省参数量。

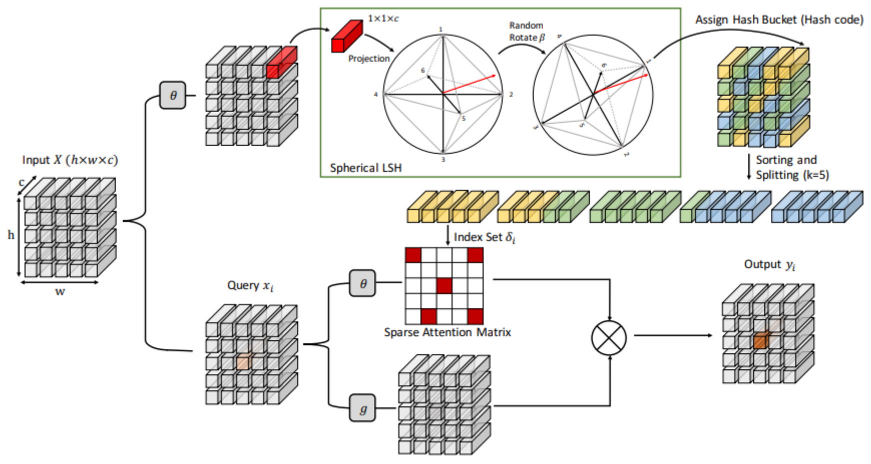
不适合超分，因为会跳过冗余信息，而冗余信息有助于超分的复原

```python
class GhostModule(nn.Module):
    def __init__(self, inp, oup, kernel_size=1, ratio=2, dw_size=3, stride=1,
relu=True):
        super(GhostModule, self).__init__()
        self.oup = oup
        init_channels = math.ceil(oup / ratio)
        new_channels = init_channels*(ratio-1)

        self.primary_conv = nn.Sequential(
            nn.Conv2d(inp, init_channels, kernel_size, stride, kernel_size//2,
bias=False),
            nn.BatchNorm2d(init_channels),
            nn.ReLU(inplace=True) if relu else nn.Sequential(),
        )

        self.cheap_operation = nn.Sequential(
            nn.Conv2d(init_channels, new_channels, dw_size, 1, dw_size//2,
groups=init_channels, bias=False),
            nn.BatchNorm2d(new_channels),
            nn.ReLU(inplace=True) if relu else nn.Sequential(),
        )

    def forward(self, x):
        x1 = self.primary_conv(x)
        x2 = self.cheap_operation(x1)
        out = torch.cat([x1,x2], dim=1)
        return out[:,:self.oup,:,:]
```

改进：GhostSR：代码未开源，未复现 https://zhuanlan.zhihu.com/p/346911265

3-Non-local Sparse Attention

[paper] [code]

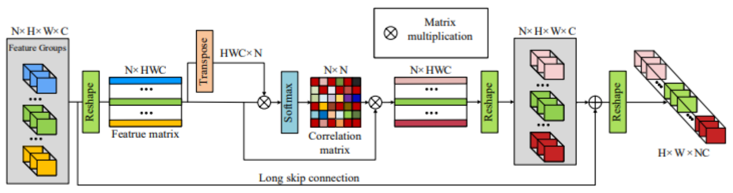


结合Non-local Attention的优点，增加了稀疏性约束迫使模块专注于相关性强和信息丰富的区域同时忽略不相关及噪声区域。

4-Layer Attention Module

[paper] [code]

尽管密集连接和跳过连接 允许将浅层信息绕过到深层，但这些操作并未利用不同层之间的相互依赖关系。 相比之下，此方法将来自每一层的特征图视为对特定类的响应，并且来自不同层的响应是相互关联的。 通过获取不同深度特征之间的依赖关系，网络可以为不同深度的特征分配不同的注意力权重，自动提高提取特征的表示能力。 因此，他们提出了一种创新的 LAM，它可以学习不同深度的特征之间的关系，从而自动提高特征表示能力。



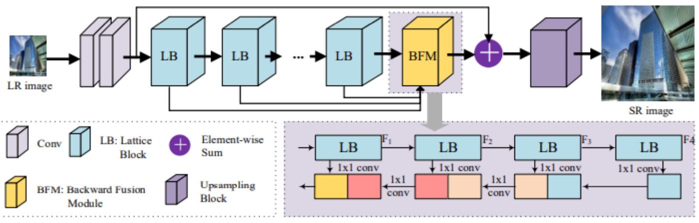
LAM模块学习来自前面模块输出的所有特征的特征相关性矩阵，利用相关性矩阵对融合到的特征加权。LAM能够增强高贡献的特征层并抑制冗余特征，且不增加额外的参数量。

```
class LAM_Module(nn.Module):
    """ Layer attention module"""
    def __init__(self, in_dim):
        super(LAM_Module, self).__init__()
        self.chanel_in = in_dim


        self.gamma = nn.Parameter(torch.zeros(1))
        self.softmax  = nn.Softmax(dim=-1)
    def forward(self,x):
        """
            inputs :
                x : input feature maps( B X N X C X H X W)
            returns :
                out : attention value + input feature
                attention: B X N X N
        """
        m_batchsize, N, C, height, width = x.size()
        proj_query = x.view(m_batchsize, N, -1)
        proj_key = x.view(m_batchsize, N, -1).permute(0, 2, 1)
        energy = torch.bmm(proj_query, proj_key)
        energy_new = torch.max(energy, -1, keepdim=True)[0].expand_as(energy)-
energy
        attention = self.softmax(energy_new)
        proj_value = x.view(m_batchsize, N, -1)

        out = torch.bmm(attention, proj_value)
        out = out.view(m_batchsize, N, C, height, width)

        out = self.gamma*out + x
        out = out.view(m_batchsize, -1, height, width)
        return out
```

5-LatticeNet

[paper] [code]

融合模块：Backward Fusion Model



通过这种反向连接的方式，BFM可以整合来自所有LB模块的特征，有助于提取更多层次的上下文信息

蝶形模块：Lattice Block

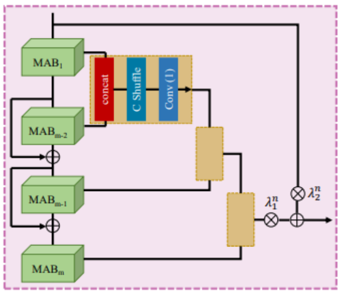
有效减少baseline模型约一半的参数量，计算复杂度也较低，性能损失较小；通过残差块的可学习组合系数与注意力机制自适应地组合，提升重要通道的权重以获得更好的SR性能。

```
## Combination Coefficient
class CC(nn.Module):
    def __init__(self, channel, reduction=16):
        super(CC, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.conv_mean = nn.Sequential(
                nn.Conv2d(channel, channel // reduction, 1, padding=0,
bias=True),
                nn.ReLU(inplace=True),
                nn.Conv2d(channel // reduction, channel, 1, padding=0,
bias=True),
                nn.Sigmoid()
        )
        self.conv_std = nn.Sequential(
                nn.Conv2d(channel, channel // reduction, 1, padding=0,
bias=True),
                nn.ReLU(inplace=True),
                nn.Conv2d(channel // reduction, channel, 1, padding=0,
bias=True),
                nn.Sigmoid()
        )

    def forward(self, x):
        # mean
        ca_mean = self.avg_pool(x)
        ca_mean = self.conv_mean(ca_mean)

        # std
        m_batchsize, C, height, width = x.size()
        x_dense = x.view(m_batchsize, C, -1)
        ca_std = torch.std(x_dense, dim=2, keepdim=True)
        ca_std = ca_std.view(m_batchsize, C, 1, 1)
        ca_var = self.conv_std(ca_std)
        cc = (ca_mean + ca_var)/2.0
        return cc


class LatticeBlock(nn.Module):
    def __init__(self, nFeat, nDiff, nFeat_slice):
        super(LatticeBlock, self).__init__()

        self.D3 = nFeat
        self.d = nDiff
        self.s = nFeat_slice

        block_0 = []
        block_0.append(nn.Conv2d(nFeat, nFeat-nDiff, kernel_size=3, padding=1,
bias=True))
        block_0.append(nn.LeakyReLU(0.05))
        block_0.append(nn.Conv2d(nFeat-nDiff, nFeat-nDiff, kernel_size=3,
padding=1, bias=True))
        block_0.append(nn.LeakyReLU(0.05))
        block_0.append(nn.Conv2d(nFeat-nDiff, nFeat, kernel_size=3, padding=1,
bias=True))
        block_0.append(nn.LeakyReLU(0.05))
        self.conv_block0 = nn.Sequential(*block_0)

        self.fea_ca1 = CC(nFeat)
        self.x_ca1 = CC(nFeat)

        block_1 = []
        block_1.append(nn.Conv2d(nFeat, nFeat-nDiff, kernel_size=3, padding=1,
bias=True))
        block_1.append(nn.LeakyReLU(0.05))
        block_1.append(nn.Conv2d(nFeat-nDiff, nFeat-nDiff, kernel_size=3,
padding=1, bias=True))
        block_1.append(nn.LeakyReLU(0.05))
        block_1.append(nn.Conv2d(nFeat-nDiff, nFeat, kernel_size=3, padding=1,
bias=True))
        block_1.append(nn.LeakyReLU(0.05))
        self.conv_block1 = nn.Sequential(*block_1)

        self.fea_ca2 = CC(nFeat)
        self.x_ca2 = CC(nFeat)

        self.compress = nn.Conv2d(2 * nFeat, nFeat, kernel_size=1, padding=0,
bias=True)

    def forward(self, x):
        # analyse unit
        x_feature_shot = self.conv_block0(x)
        fea_ca1 = self.fea_ca1(x_feature_shot)
        x_ca1 = self.x_ca1(x)

        p1z = x + fea_ca1 * x_feature_shot
        q1z = x_feature_shot + x_ca1 * x

        # synthes_unit
        x_feat_long = self.conv_block1(p1z)
        fea_ca2 = self.fea_ca2(q1z)
        p3z = x_feat_long + fea_ca2 * q1z
        x_ca2 = self.x_ca2(x_feat_long)
        q3z = q1z + x_ca2 * x_feat_long

        out = torch.cat((p3z, q3z), 1)
        out = self.compress(out)

        return out
```

PS：实际加入到我们的模型中，BFM涨点0.01dB左右，LB模块会掉0.1dB的点

6-MFFG module

[paper] [code]



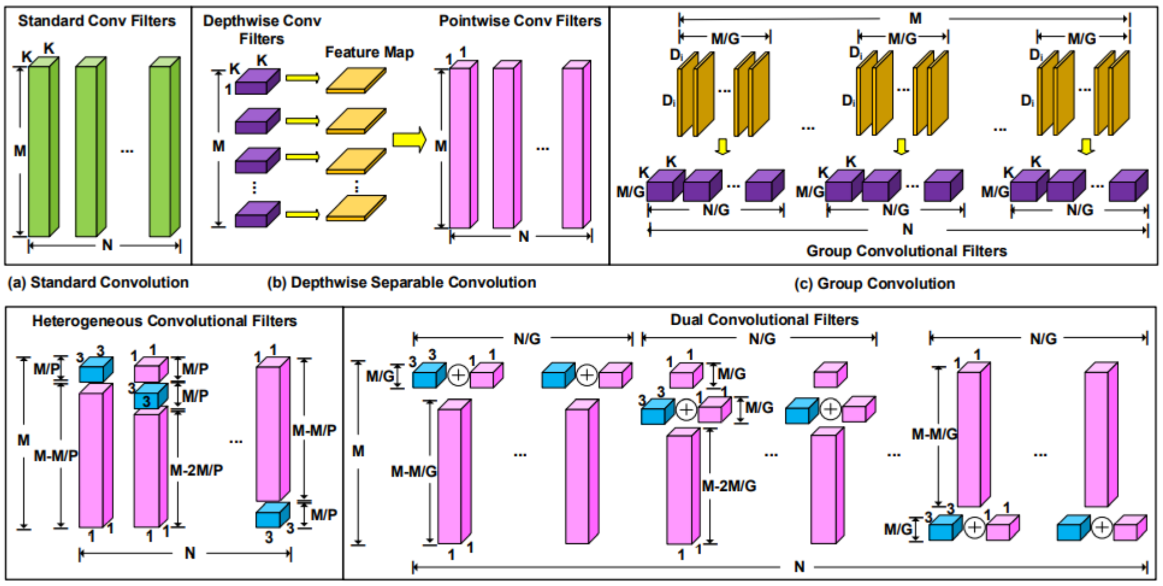
M-FFG中，每个MAB提取到的特征除了密集连接和长连接，融合方式也不单纯是concat，还包括通道混洗和1×1卷积，充分融合通道间的信息。

```
class FFG(nn.Module):
    def __init__(
        self, n_feats, wn, act=nn.ReLU(True)):
        super(FFG, self).__init__()

        self.b0 = MAB(n_feats=n_feats,reduction_factor=4)
        self.b1 = MAB(n_feats=n_feats,reduction_factor=4)
        self.b2 = MAB(n_feats=n_feats,reduction_factor=4)
        self.b3 = MAB(n_feats=n_feats,reduction_factor=4)

        self.reduction1 = wn(nn.Conv2d(n_feats*2, n_feats, 1))
        self.reduction2 = wn(nn.Conv2d(n_feats*2, n_feats, 1))
        self.reduction3 = wn(nn.Conv2d(n_feats*2, n_feats, 1))
        self.res_scale = Scale(1)
        self.x_scale = Scale(1)

    def forward(self, x):
        x0 = self.b0(x)
        x1 = self.b1(x0)+x0
        x2 = self.b2(x1)+x1
        x3 = self.b3(x2)
        res1 = self.reduction1(channel_shuffle(torch.cat([x0, x1],dim=1), 2))
        res2 = self.reduction2(channel_shuffle(torch.cat([res1, x2], dim=1), 2))
        res = self.reduction3(channel_shuffle(torch.cat([res2,x3], dim=1), 2))

        return self.res_scale(res) + self.x_scale(x)
```

PS：采用M-FFG融合我们的网络模块，效果还是比较显著

7-DualConv

[paper]



(a) Standard Convolution (b) Depthwise Separable Convolution (c) Group Convolution
(d) Heterogeneous Convolution (e) Dual Convolution

不仅解决分组卷积中分组间的信息不互通，且相比HetConv提升了深度神经网络的性能

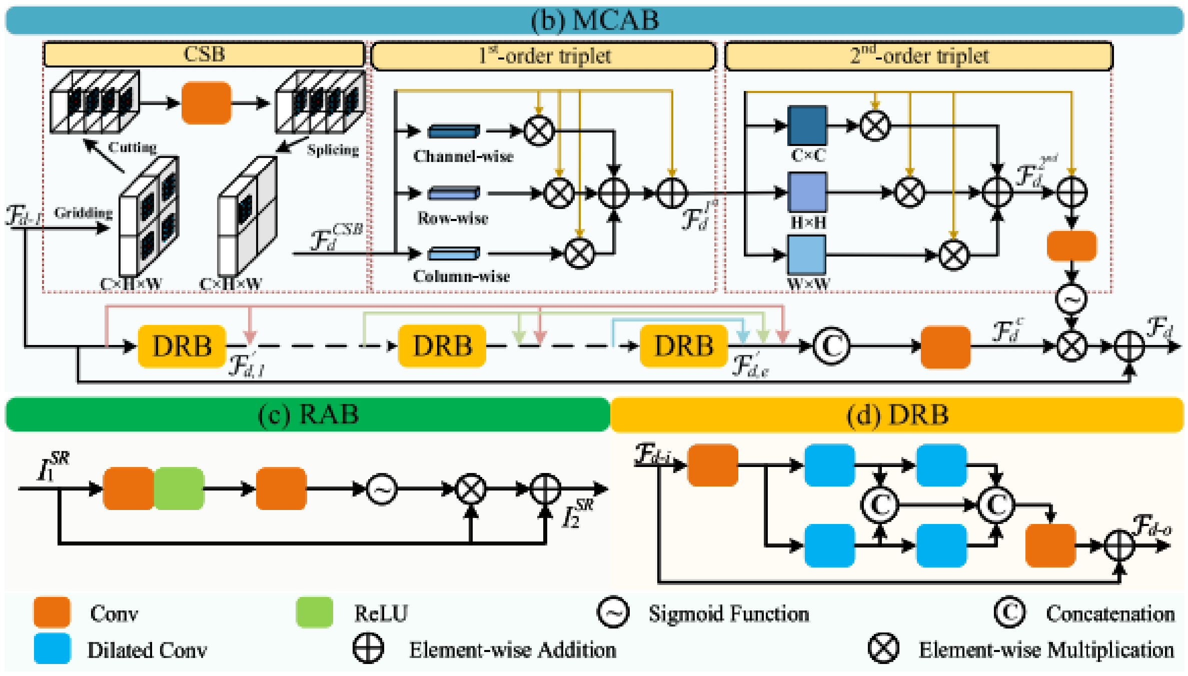
PS：代码未开源，参考HetConv的复现代码重写了一个，能够减小参数量，但处理时间增加了两倍

```
class Efficient_DualConv2d(nn.Module):
    def __init__(self, in_feats, out_feats, p=4):
        super(Efficient_DualConv2d, self).__init__()
        if in_feats % p != 0:
            raise ValueError('in_channels must be divisible by p')
        if out_feats % p != 0:
            raise ValueError('out_channels must be divisible by p')
        self.conv3x3 = nn.Conv2d(in_feats, out_feats, kernel_size=3, padding=1,
groups=p)
        self.conv1x1 = nn.Conv2d(in_feats, out_feats, kernel_size=1)

    def forward(self, x):
        return self.conv3x3(x) + self.conv1x1(x)
```

但复现得不太对，尽管参数量下降了

8-multi-context attentive block

[paper] [code]



模块包括上下文特征提取分支（有多个DRB堆叠而成）和注意力分支。

在上下文特征提取分支中，以扩张残差块（Dilated Residual Block，DRB）为基本单元，通过扩大感受野来探索更多的上下文线索，同时提取具有不同上下文特征的特征，用于重建视觉愉悦的 HR 图像。

注意力分支包括一个cutting-splicing block、1阶triplet、2阶triplet。为了同时捕获局部块和全局图的空间依赖关系，论文提出了CSB来提取局部模式并利用全局区域中丰富的结构线索和自相似性。每个注意力三元组由三个分支组成，它们分别负责捕获输入张量的 (C, H)、(C, W) 和 (H, W) 维度之间的跨维度交互。 通过利用通道维度和空间维度之间的相互依赖关系，网络可以有效地专注于信息丰富的上下文特征。

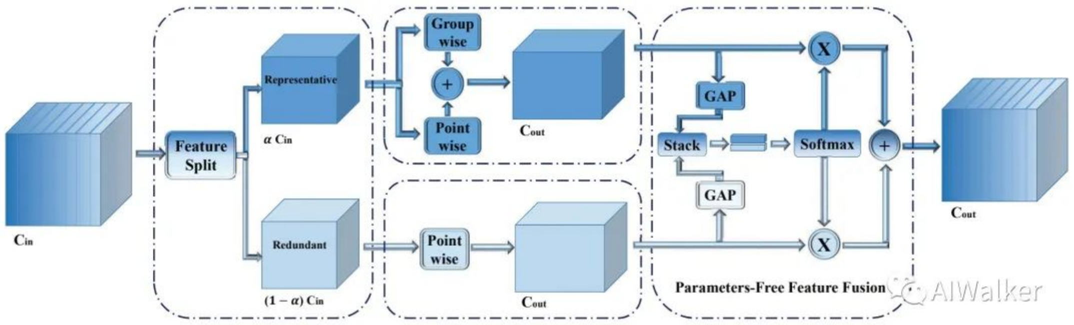
```
class CSB(nn.Module):
    def __init__(self, n_feats):
        super(CSB, self).__init__()
        self.conv1 = nn.Conv2d(n_feats*4, n_feats*4, kernel_size=1, padding=0,
bias=False)

    def forward(self, x):
        # print(x.size(2)//2)
        f1 = x[:, : , 0:x.size(2)//2, 0:x.size(3)//2]
        f2 = x[:, : , x.size(2)//2:, 0:x.size(3)//2]
        f3 = x[:, : , 0:x.size(2)//2, x.size(3)//2:]
        f4 = x[:, : , x.size(2)//2:, x.size(3)//2:]
        # print(f1.size(), f2.size(), f3.size(), f4.size())
        f_all = torch.cat([f1,f2,f3,f4], 1)
        # print(f_all.size())
        f_all = self.conv1(f_all)
        out = x
        out[:,:,0:x.size(2)//2, 0:x.size(3)//2] =
f_all[:,0:f_all.size(1)//4,:,:]
        out[:,:,x.size(2)//2:, 0:x.size(3)//2] =
f_all[:,f_all.size(1)//4:f_all.size(1)//2,:,:]
        out[:,:,0:x.size(2)//2, x.size(3)//2:] =
f_all[:,f_all.size(1)//2:f_all.size(1)*3//4,:,:]
        out[:,:,x.size(2)//2:, x.size(3)//2:] = f_all[:,f_all.size(1)*3//4:,:,:]
        return out

class First_Order(nn.Module):
    def __init__(self):
        super(First_Order, self).__init__()
        self.adaptive = nn.AdaptiveAvgPool2d((1,1))

    def forward(self, x):
        first_c = F.sigmoid(self.adaptive(x))
        first_c = first_c * x
        xh = x.permute(0,2,1,3)
        first_h = F.sigmoid(self.adaptive(xh))
        first_h = (first_h * xh).permute(0,2,1,3)
        xw = x.permute(0,3,2,1)
        first_w = F.sigmoid(self.adaptive(xw))
        first_w = (first_w * xw).permute(0,3,2,1)

        return first_c + first_h + first_w + x

class Second_Order(nn.Module):
    def __init__(self):
        super(Second_Order, self).__init__()

    def forward(self, x):
        second_c = F.sigmoid(torch.mean(x, 1).unsqueeze(1))
        second_c = second_c * x
        xh = x.permute(0,2,1,3)
        second_h = F.sigmoid(torch.mean(xh, 1).unsqueeze(1))
        second_h = (second_h * xh).permute(0,2,1,3)
        xw = x.permute(0,3,2,1)
        second_w = F.sigmoid(torch.mean(xw, 1).unsqueeze(1))
        second_w = (second_w * xw).permute(0,3,2,1)

        return second_c + second_h + second_w + x

class MCAB(nn.Module):
    def __init__(self, n_channels, n_denselayer=6, n_feats=64):
        super(MCAB, self).__init__()
        self.drb = DRB(n_channels, n_denselayer, n_feats)
        self.csb = CSB(n_feats)
        self.first = First_Order()
        self.second = Second_Order()
        self.conv3 = nn.Conv2d(n_feats, n_feats, kernel_size=3, padding=1,
bias=False)

    def forward(self, x):
        down_b = self.drb(x)
        top_b = self.csb(x)
        top_b = self.second(self.first(top_b))
        top_b = F.sigmoid(self.conv3(top_b))
        return down_b*top_b
```

PS：DRB扩大感受野的同时，参数量和memory也增加了许多。注意力分支中CSB有效果但只能处理(分辨率可以被4整除)的图片，推理阶段需要补零。

9-SPConv

[paper] [code]

同一层内的许多特征具有相似却不平等的表现模式。然而，这类具有相似模式的特征却难以判断是否存在冗余或包含重要的细节信息。因此，不同于直接移除不确定的冗余特征方案，作者提出了一种基于Split的卷积计算单元(称之为SPConv)，它运行/训练存在相似模型冗余且仅需非常少的计算量。首先，将输入特征拆分为representative部分与uncertain部分；然后，对于representative部分特征采用相对多的计算复杂度操作提取重要信息，对于uncertain部分采用轻量型操作提取隐含信息；最后，为重新校准与融合两组特征，作者采用了无参特征融合模块。

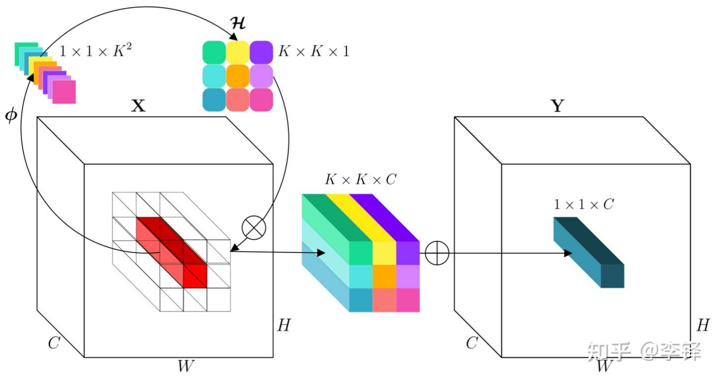


PS：替换普通卷积后，参数量下降很可观，但是memory占用很大，推理速度也较慢

10-involution

[paper] [code]

在通道维度共享kernel，而在空间维度采用空间特异的kernel进行更灵活的建模。



针对输入feature map的一个坐标点上的特征向量，先通过 ϕ (FC-BN-ReLU-FC)和reshape (channel-to-space)变换展开成kernel的形状，从而得到这个坐标点上对应的involution kernel，再和输入feature map上这个坐标点邻域的特征向量进行Multiply-Add得到最终输出的feature map。

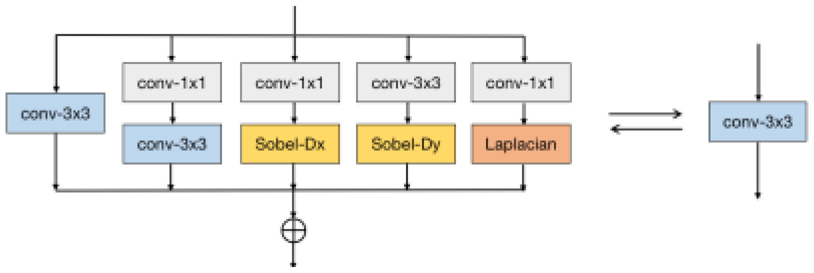
一些待探索的坑：

1. 关于广义的involution中kernel生成函数空间进一步的探索；
2. 类似于deformable convolution加入offest生成函数，使得这个op空间建模能力的灵活性进一步提升；
3. 结合NAS的技术搜索convolution-involution混合结构（原文Section 4.3）

PS：不能直接用involution替换普通卷积，会出现loss变为inf的情况，但可以用involution+convolution的模式

11-Edge-oriented covolution block

[paper] [code]



考虑到：

- 更宽的特征可以显著提升网络表达能力和超分性能
- 边缘信息对超分任务很有用

提出了边缘引导的ECB，推理阶段等价为一个普通卷积。最大亮点是将结构重参数思想与low-level领域结合。同时巧妙地将一阶梯度和二阶梯度结合，采用Laplacian滤波器提取二阶梯度（更稳定，对噪声更鲁棒）

PS：替换普通卷积后，训练前期提取特征的能力不错，但后继乏力，对参数量大的模型应该有效果

12-Knowlege Distillation

[paper] [code]

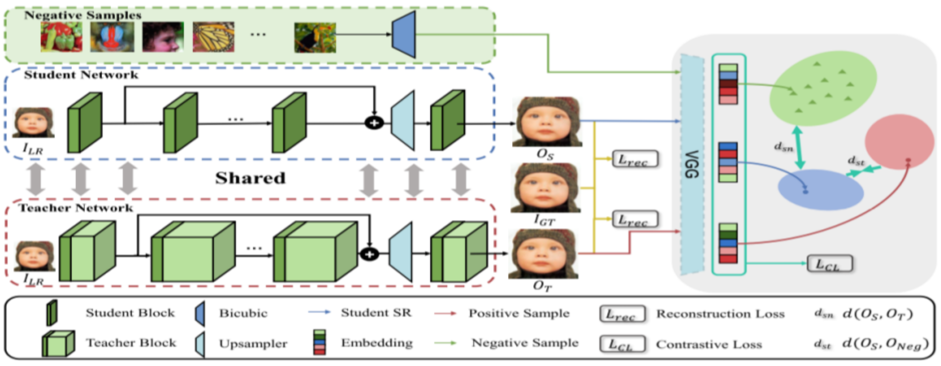


Figure 2: The framework of our proposed CSD. Here we choose EDSR as our backbone for example. The darker green parts of the CSSR-Net (student) are shared from its teacher. The student and teacher separately produce a high-resolution image constrained by the reconstruction loss. The rich knowledge is constructed by contrastive loss and explicitly transferred from teacher to student, where the output O_S is pulled to closer to the O_T and pushed far away from the negative samples in the embedding space of VGG network.

对比自蒸馏框架：CSSR-Net其实可以是任何超分网络的子网络，与原网络的区别仅在于channel数量上，在CSD框架中作为学生网络，来作为加速优化的目标产物模型。在CSD框架中，CSSR-Net被耦合进教师网络，共享教师网络的部分参数（即共同参数部分）。损失函数引入了对比学习损失。

PS：降低参数量很显著，但是学生和教师网络联合训练的方式，导致学生和教师网络涨点都很慢

其他

ELAN：自注意力、移位卷积、共享注意力机制 [paper] [code]

VAN：使自注意力的自适应和长距离相关 [paper] [code]

APBN：8-bit量化版高效网络、量化感知训练策略 [paper] [code]

调参

炼丹技巧

1. 深度学习模型多loss调参技巧

- GradNorm
- Multi-Task Learning as Multi-Objective Optimization
- Multi-task likelihoods
- 玄学调参

2. 升级版炼丹手法助力RCAN性能媲美SwinIR

- Longer Training：采用更长的训练周期以缓解欠拟合问题
- Large-patch Finetuning：设计了一种两阶段训练策略：首先采用标准块进行训练，然后采用更大的块进行微调。
- Low-precision Training：已有研究表明：采用FP16进行图像分类模型训练可以保持、甚至轻微改善模型性能，同时大幅降低训练时间与GPU占用。但是，我们发现：FP16训练在图像超分中具有截然不同的行为表现。
- Regularization Technique：尝试了图像分类中广泛采用的正则化技术，比如Mixup、随机深度等。然而，由于图像RCAN存在欠拟合而非过拟合问题，故RCAN并未从正则技术中受益。
- Warm Start：对于不同尺度的图像超分模型，我们采用x2预训练模型对x3和x4进行初始化，称之为warm start。

3. 优化器采用AdamP，Set5数据集上涨点0.1dB

4. 数据增强：比较常见的是对图片进行翻转平移、旋转操作，但也有提出其他策略的

5. torch中manual_seed的作用

6. 机器学习中的超参数优化

7. NAS网络架构搜索

模型复杂度统计

- 官方统计参数量、Flops、Activations的方法 refer

  ```
  from utils.utils_modelsummary import get_model_activation, get_model_flops
  input_dim = (3, 256, 256)  # set the input dimension

  activations, num_conv2d = get_model_activation(model, input_dim)
  logger.info('{:>16s} : {:<.4f} [M]'.format('#Activations',
  activations/10**6))
  logger.info('{:>16s} : {:<d}'.format('#Conv2d', num_conv2d))

  flops = get_model_flops(model, input_dim, False)
  logger.info('{:>16s} : {:<.4f} [G]'.format('FLOPs', flops/10**9))

  num_parameters = sum(map(lambda x: x.numel(), model.parameters()))
  logger.info('{:>16s} : {:<.4f} [M]'.format('#Params', num_parameters/10**6))
  ```

- 其他python库：torchprofile, torchstat, torchsummary

模型可视化

1. 深度学习训练过程可视化

2. pytorch中使用Tensorboard可视化训练过程