20210313

▼ 论文：CVPR2021 Inverting the Inherence of Convolution for Visual Recognition, a brand new neural operator

代码：https://github.com/d-li14/involution

一下代码来源于：
https://github.com/ultralytics/yolov5/pull/2435/commits/55667800000d9ae3b90d88c1283e7690d0d14824

```
class Involution(nn.Module):
    # Involution module https://arxiv.org/abs/2103.06255
    def init(self, ch, k, s):  # ch_in, kernel, stride
    super(Involution, self).init()
    reduction_ratio = 4
    self.gch = 16  # group channels
    self.nk = k ** 2  # number of kernel elements
    self.stride = s
    self.c1 = ch
    c2 = ch // reduction_ratio
    self.groups = self.c1 // self.gch
    self.cv1 = Conv(c1=ch, c2=c2, k=1)
    self.cv2 = nn.Conv2d(c2, k ** 2 * self.groups, 1)
    if s > 1:
        self.avgpool = nn.AvgPool2d(s, s)
    self.unfold = nn.Unfold(k, 1, (k - 1) // 2, s)

def forward(self, x):
    weight = self.cv2(self.cv1(x if self.stride == 1 else self.avgpool(x)))
    b, c, h, w = weight.shape  # BCHW
    weight = weight.view(b, self.groups, 1, self.nk, h, w)
    x = self.unfold(x).view(b, self.groups, self.gch, self.nk, h, w)
    return (weight * x).sum(3).view(b, self.c1, h, w)
```

▼ ASFF

论文：Learning Spatial Fusion for Single-Shot Object Detection

代码：https://github.com/ruinmessi/ASFF?utm_source=catalyzex.com

代码来源于：https://github.com/positive666/yolov5/blob/master/models/common.py

```
class ASFFV5(nn.Module):
    def **init**(self, level,out,multiplier=1, rfb=False, vis=False, act_cfg=True):
        """
        ASFF version for YoloV5 .
        different than YoloV3
        multiplier should be 1, 0.5
        which means, the channel of ASFF can be
        512, 256, 128 -> multiplier=1
        256, 128, 64 -> multiplier=0.5
        For even smaller, you need change code manually.
        """
        super(ASFFV5, self).**init**()
        self.level = level
        self.dim = [int(1024*multiplier), int(512*multiplier),
        int(256*multiplier)]
        # print(self.dim)
        self.inter_dim = self.dim[self.level]
        if level == 0:
            self.stride_level_1 = Conv(int(512*multiplier), self.inter_dim, 3, 2)

            self.stride_level_2 = Conv(int(256*multiplier), self.inter_dim, 3, 2)

            self.expand = Conv(self.inter_dim, int(
                1024*multiplier), 3, 1)
        elif level == 1:
            self.compress_level_0 = Conv(
                int(1024*multiplier), self.inter_dim, 1, 1)
            self.stride_level_2 = Conv(
                int(256*multiplier), self.inter_dim, 3, 2)
            self.expand = Conv(self.inter_dim, int(512*multiplier), 3, 1)
        elif level == 2:
            self.compress_level_0 = Conv(
                int(1024*multiplier), self.inter_dim, 1, 1)
            self.compress_level_1 = Conv(
                int(512*multiplier), self.inter_dim, 1, 1)
            self.expand = Conv(self.inter_dim, int(
                256*multiplier), 3, 1)

        # when adding rfb, we use half number of channels to save memory
        compress_c = 8 if rfb else 16
        self.weight_level_0 = Conv(
            self.inter_dim, compress_c, 1, 1)
        self.weight_level_1 = Conv(
            self.inter_dim, compress_c, 1, 1)
        self.weight_level_2 = Conv(
            self.inter_dim, compress_c, 1, 1)

        self.weight_levels = Conv(
            compress_c*3, 3, 1, 1)
        self.vis = vis

    def forward(self, x): #l,m,s
        """
        # 128, 256, 512
        512, 256, 128
        from small -> large
        """
        x_level_0=x[2] #l
        x_level_1=x[1] #m
        x_level_2=x[0] #s
        # print('x_level_0: ', x_level_0.shape)
        # print('x_level_1: ', x_level_1.shape)
        # print('x_level_2: ', x_level_2.shape)
        if self.level == 0:
            level_0_resized = x_level_0
            level_1_resized = self.stride_level_1(x_level_1)
            level_2_downsampled_inter = F.max_pool2d(
                x_level_2, 3, stride=2, padding=1)
            level_2_resized = self.stride_level_2(level_2_downsampled_inter)
        elif self.level == 1:
            level_0_compressed = self.compress_level_0(x_level_0)
            level_0_resized = F.interpolate(
                level_0_compressed, scale_factor=2, mode='nearest')
            level_1_resized = x_level_1
            level_2_resized = self.stride_level_2(x_level_2)
        elif self.level == 2:
            level_0_compressed = self.compress_level_0(x_level_0)
            level_0_resized = F.interpolate(
                level_0_compressed, scale_factor=4, mode='nearest')
            x_level_1_compressed = self.compress_level_1(x_level_1)
            level_1_resized = F.interpolate(
                x_level_1_compressed, scale_factor=2, mode='nearest')
            level_2_resized = x_level_2

        # print('level: {}, l1_resized: {}, l2_resized: {}'.format(self.level,
        #      level_1_resized.shape, level_2_resized.shape))
        level_0_weight_v = self.weight_level_0(level_0_resized)
        level_1_weight_v = self.weight_level_1(level_1_resized)
        level_2_weight_v = self.weight_level_2(level_2_resized)
        # print('level_0_weight_v: ', level_0_weight_v.shape)
        # print('level_1_weight_v: ', level_1_weight_v.shape)
        # print('level_2_weight_v: ', level_2_weight_v.shape)

        levels_weight_v = torch.cat(
            (level_0_weight_v, level_1_weight_v, level_2_weight_v), 1)
        levels_weight = self.weight_levels(levels_weight_v)
        levels_weight = F.softmax(levels_weight, dim=1)
        fused_out_reduced = level_0_resized * levels_weight[:, 0:1, :, :] +\\
            level_1_resized * levels_weight[:, 1:2, :, :] +\\
            level_2_resized * levels_weight[:, 2:, :, :]

        out = self.expand(fused_out_reduced)

        if self.vis:
            return out, levels_weight, fused_out_reduced.sum(dim=1)
        else:
            return out
```