Supplementary Materials for "ParC-Net: Position Aware Circular Convolution with Merits from ConvNets and Transformer"

1 The implementation of ParC block

ParC block, implemented in PyTorch

```
class ParC_block(BaseModule):
    def __init__(self,
                 dim: int,
                 base_kernel_size: int,
                 instance_kernel_method='crop',
                 use_pe:Optional[bool]=True,
                 ffn_dim: Optional[int]=2,
                 ffn_dropout=0.0,
                 dropout=0.1):

        super(EdgeFormer_block, self).__init__()

        bk_size = base_kernel_size
        # token mixer,
        self.pre_Norm_1 = nn.BatchNorm2d(num_features=dim)
        self.pre_Norm_2 = nn.BatchNorm2d(num_features=dim)

        self.bk_1_H = nn.Conv2d(dim, dim, (bk_size, 1), groups=dim).weight
        self.bk_1_W = nn.Conv2d(dim, dim, (1, bk_size), groups=dim).weight
        self.bk_2_H = nn.Conv2d(dim, dim, (bk_size, 1), groups=dim).weight
        self.bk_2_W = nn.Conv2d(dim, dim, (1, bk_size), groups=dim).weight

        self.instance_kernel_method = instance_kernel_method

        if use_pe:
            self.base_pe_1_H = nn.Parameter(torch.randn(1, dim, bk_size, 1))
            self.base_pe_1_W = nn.Parameter(torch.randn(1, dim, 1, bk_size))
            self.base_pe_2_H = nn.Parameter(torch.randn(1, dim, bk_size, 1))
            self.base_pe_2_W = nn.Parameter(torch.randn(1, dim, 1, bk_size))

        self.use_pe = use_pe
        self.dim = dim

        # channel mixer
        self.ffn = nn.Sequential(
            nn.BatchNorm2d(num_features=2*dim),
            nn.Conv2d(2*dim, ffn_dim, kernel_size=(1, 1), bias=True),
```

```
            nn.Hardswish(),
            Dropout(p=ffn_dropout),
            nn.Conv2d(ffn_dim, 2*dim, kernel_size=(1, 1), bias=True),
            Dropout(p=dropout)
        )

        self.ca = CA_layer(channel=2*dim)

    def get_instance_kernel(self, instance_kernel_size):
        ik_size = instance_kernel_size
        H_shape = [ik_size, 1]
        W_shape = [1, ik_size]

        return F.interpolate(self.bk_1_H, H_shape, mode='bilinear',
                              align_corners=True), \
               F.interpolate(self.bk_1_W, W_shape, mode='bilinear',
                              align_corners=True), \
               F.interpolate(self.bk_2_H, H_shape, mode='bilinear',
                              align_corners=True), \
               F.interpolate(self.bk_2_W, W_shape, mode='bilinear',
                              align_corners=True),

    def get_instance_pe(self, instance_kernel_size):
        ik_size = instance_kernel_size

        return \
            F.interpolate(self.base_pe_1_H, [ik_size, 1], mode='bilinear',
                align_corners=True).expand(1, self.dim, ik_size, ik_size), \
            F.interpolate(self.base_pe_1_W, [1, ik_size], mode='bilinear',
                align_corners=True).expand(1, self.dim, ik_size, ik_size), \
            F.interpolate(self.base_pe_2_H, [ik_size, 1], mode='bilinear',
                align_corners=True).expand(1, self.dim, ik_size, ik_size), \
            F.interpolate(self.base_pe_2_W, [1, ik_size], mode='bilinear',
                align_corners=True).expand(1, self.dim, ik_size, ik_size)


    def forward(self, x: Tensor) -> Tensor:

        x_1, x_2 = torch.chunk(x, 2, 1)
        x_1_res, x_2_res = x_1, x_2
        _, _, f_s, _ = x_1.shape

        K_1_H, K_1_W, K_2_H, K_2_W = self.get_instance_kernel(f_s)

        if self.use_pe:
            pe_1_H, pe_1_W, pe_2_H, pe_2_W = self.get_instance_pe(f_s)

        # # token mixer
        # stage 1
        if self.use_pe:
```

```
    x_1, x_2 = x_1 + pe_1_H, x_2 + pe_1_W
x_1, x_2 = self.pre_Norm_1(x_1), self.pre_Norm_2(x_2)
x_1_1 = F.conv2d(torch.cat((x_1, x_1[:, :, :-1, :]), dim=2),
                 weight=K_1_H, padding=0, groups=self.dim)
x_2_1 = F.conv2d(torch.cat((x_2, x_2[:, :, :, :-1]), dim=3),
                 weight=K_1_W, padding=0, groups=self.dim)

# stage 2
if self.use_pe:
    x_1_1, x_2_1 = x_1_1 + pe_2_W, x_2_1 + pe_2_H
x_1_2 = F.conv2d(torch.cat((x_1_1, x_1_1[:, :, :, :-1]), dim=3),
                 weight=K_2_W, padding=0, groups=self.dim)
x_2_2 = F.conv2d(torch.cat((x_2_1, x_2_1[:, :, :-1, :]), dim=2),
                 weight=K_2_H, padding=0, groups=self.dim)
x_1, x_2 = x_1_res + x_1_2, x_2_res + x_2_2

## channel mixer
x_ffn = torch.cat((x_1, x_2), dim=1)
x_ffn = x_ffn + self.ca(self.ffn(x_ffn))

return x_ffn
```

2 Training settings

Training settings of image classification, object detection and semantic segmentation are listed in Table 1.

Table 1. Training settings

Tasks	Image classification	Object detection	Semantic segmentation
Pretraining set	None	ImageNet-1K	ImageNet-1K
Training set	ImageNet-1K	MS-COCO	MS-COCO + PASCAL VOC
Validation set	ImageNet-1K	MS-COCO	PASCAL VOC
Optimizer	AdamW, β_1=0.9, β_2=0.999	AdamW, β_1=0.9, β_2=0.999	AdamW, β_1=0.9, β_2=0.999
Weight decay	0.025	0.025	0.025
Maximum learning rate	4e-3	1e-6	1.e-6
Minimum learning rate	4e-4	9e-4	9e-4
Learning rate schedule	consine	consine	cosine
Num of GPUs	8	4	4
Batch size per GPU	128	32	32
Warmup iterations	3000	500	500
Training epochs	300	200	50
Random crop	1.0	1.0	1.0
Random flip	0.5	0.5	0.5
Multi-scale sampler	160-320	None	384-768
Label smoothing	0.1	0.1	0.1
EMA	0.9995	0.9995	0.9995

Table 2. Model scaling experiments

Tiny			Small			Large		
Models	Scale	Top1	Models	Scale	Top1	Models	Scale	Top1
DeiT-T	5.7	72.2	ConvNext-T (0.5×)	7.4	77.5	Swin-T	28	81.3
MobileViT-XS	2.3	74.8	MobileViT-S	5.6	78.4	ConvNext-T	29	**82.1**
ParC-Net-XS	**2.1**	**75.0**	ParC-Net-S	**5.0**	**78.6**	ParC-Net-H	**20**	81.9

3 Model scaling experiments

To evaluate the scalability of the proposed ParC-Net, we construct three models of different sizes, including ParC-Net-XS, ParC-Net-S and ParC-Net-H (huge). Experimental results are listed in Table 3. From Table 3, we can get two conclusions: 1) ParC-Net models achieves best performance when constraining models as light-weight models. ParC-Net-XS and ParC-Net-S achieves best performance while having fewer parameters than their counterparts. The experimental results is consistent with the purpose of improving light-weight ConvNets; 2) ParC-Net achieves comparable results when scaling up models to large size. Compared with Swin-T, ParC-Net-H contains 71% parameters while achieving 0.6% higher accuracy. Compared with most recent released work ConvNext, ParC-Net-H saves 31% parameters and has comparable accuracy.