动态轻量级高分辨率人体姿态估计网络

2021 级　信息与通信工程　　谢唯嘉　　6120210299

摘要：高分辨率网络在提取多尺度特征用于人体姿态估计方面表现出了出色的能力，但不能捕获人体关节点之间的网络中远距离相互作用，而且计算复杂度较高。为了解决这些问题，我们提出了一个动态轻量级高分辨率网络，可以高效地提取多尺度上下文信息和建立人体姿态估计的长期空间依赖性模型。具体地，我们提出了动态分割卷积和自适应上下文建模两种方法，并将它们嵌入到两个新颖的轻量级块。

关键点：人体姿态估计; 深度学习; 计算机视觉

1　引言

近些年来，随着深度学习的发展，人体姿态估计在单帧图片 [1] [2] [3] [4] 和视频 [5] [6] 中的单人姿态估计以及单眼图像中的多人姿态估计 [7] [8] 中取得了显著的进展，人体姿态估计广泛应用于人机交互、游戏、虚拟现实、视频监控、运动分析、医疗辅助等领域，是计算机视觉领域非常热门的研究课题。这一进展得益于基于深度学习的体系结构的使用 [9] [10] 和大规模基准数据集的可用性，如“MPII Human Pose”[11] 和“MS COCO”[12]。重要的是，这些基准数据集不仅为基于深度学习的方法的训练提供了广泛的训练集，而且还建立了详细的指标，用于在众多竞争的方法之间进行直接和公平的性能比较。

最近的研究 [13] 已经证明了长期空间依赖性在人体姿态估计中的好处。长程空间相关性的一般概念是在大视场范围内对空间信息的全局认识。以前的高分辨率网络 [13] [14] [15] 主要依赖于并行分支中网络中深度叠加的卷积层来提取多尺度特征来构建空间相关性，但受限于输入数据通道的宽度，这会严重影响轻量级网络的容量。因此，对于高分辨率人体姿态估计网络的一个非常重要的问题是如何用更有效的方法来建模远程空间依赖性，增强轻量级高分辨率网络。

针对上述问题，我们提出了一种动态轻量级高分辨率网络，我们采用了类似于 HRNet 的架构并开发了两个动态轻量级块，以提高整体效率。首先，我们提出了一种动态分割卷积 (Dynamic Split Convolution, DSC)，它可以动态提取多尺度上下文信息，并通过两个超参数优化其容量和复杂度之间的权衡，从而比标准卷积更有效和灵活。然后，我们通过设计一种新的自适应上下文建模 (Adaptive Context Modeling, ACM) 方法，将远程空间依赖性引入到高分辨率网络中，使模型能够学习人体姿态的局部和全局模式。最后，我们将 DSC 和 ACM 嵌入到两个动态轻量级块中，这两个块是专门为高分辨率网络设计的，充分利用并行多分辨率架构，作为我们动态轻量级高分辨率人体姿态估计网络的基本组件单元。

2　网络结构

基础的网络结构是使用 HRNet [16] 作为网络骨干，并使用两个提出的动态块进行调整如图1。使用跨分辨率模块用蓝色区域标记。整体的网络结构是由一个具有最高分辨率的高分辨率主分支和三个高分辨率到低分辨率的分支组成，这些分支在每个新阶段开始时依次并行地添加到网络中。与之前添加的分支相比，每个新添加的分支具有一半的空间分辨率和两倍的通道数量。

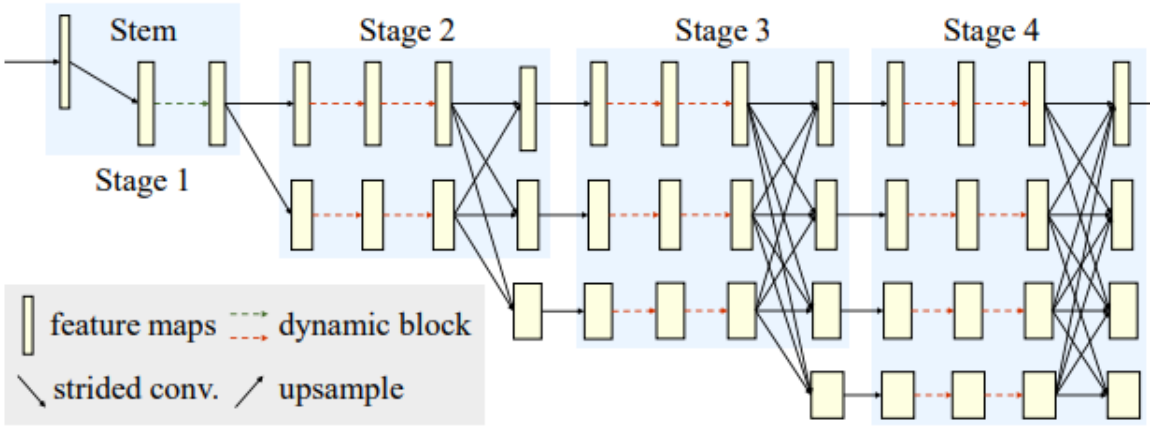

图 1. 整体网络结构

第一个阶段也称为 Stem，包含一个 3×3 的跨卷积和一个主分支上的动态全局上下文 (DGC) 块。每个后续阶段由一系列交叉解析模块组成，这些模块由两个动态多尺度上下文组成 (DMC) 块和跨所有分支交换信息的多尺度融合层。分辨率最高的主分支保持高分辨率表示，为后续的人体姿态估计提供骨干网的最终输出。

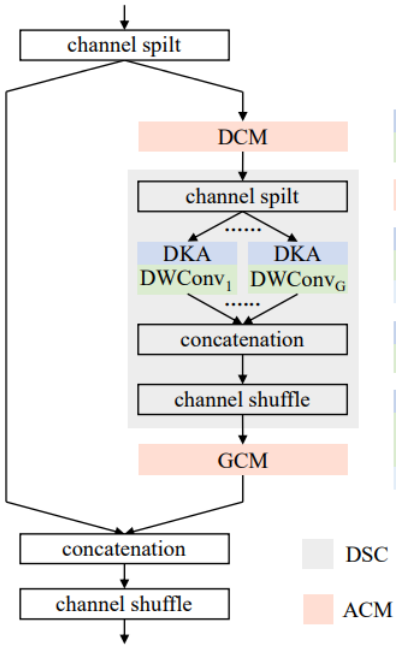

图 2. 动态多尺度上下文块的结构

图2方法中将动态分裂卷积 (Dynamic Split Convolution, DSC) 应用于动态内核聚合 (Dynamic Kernel Aggregation, DKA)DMC 块和 DGC 块中的每个卷积。DCM(密集上下文建模) 和 GCM(全局上下文建模) 是我们提出的自适应上下文建模 (ACM) 的两个实例。

DMC 块对一半图片通道应用层序列，而 DGC 块对所有两组通道应用两种不同的层序列。DMC 块中的层序列包含一个密集上下文建模 (DCM) 操作，一个 DSC 和一个全球上下文建模 (GCM)。DCM 和 GCM 都是 ACM 方法的实例。

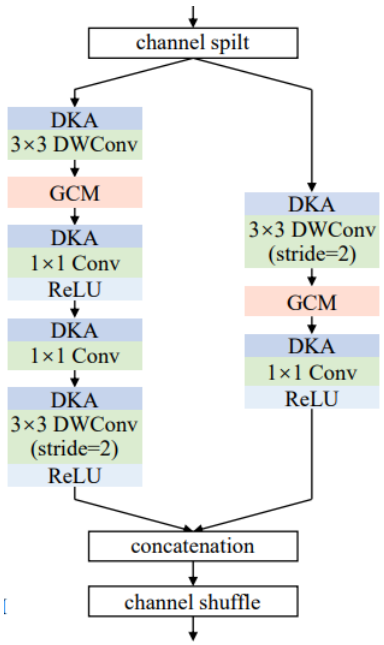

图 3. 动态全局上下文块的结构

在 DGC 块中 (如图3)，一个 3×3 阶深度卷积，一个 GCM 和一个在一组信道上进行 1×1 卷积，在另一组信道上分别进行 3×3 深度卷积、GCM 卷积、1×1 卷积和 3×3 跨步深度卷积。DGC 块和 DSC 层的每一次卷积都通过动态内核聚合 (Dynamic kernel Aggregation, DKA) 生成卷积内核。

2.1 自适应上下文建模 (ACM)

ACM 方法可以抽象为以下三个步骤:(a) 自适应上下文池化，通过 1×1 的信道池化卷积和 softmax 层建立快捷连接，形成上下文掩码，然后通过一系列分辨率自适应变换将掩码应用到特征映射上，获得空间上下文特征;(b) 上下文移位，通过两个 1×1 卷积和非线性激活将空间相关的上下文特征重新组合在一起;(c) 上下文加权，采用对应元素加权相加操作。上下文加权 (Context weighted)，是指在输入特征和移位的上下文特征之间进行元素加权运算，对对应特征的上下文关系进行建模。

ACM 方法的这种抽象可以定义为:

$$Y = Weight(X, Shift(ACPool(H,W)(X)))$$

其中 $ACPool(H,W)(X)$ 表示自适应上下文池，将输入特征 X 池到特定的输出大小 $H\times W$，$Shift(.)$ 表示上下文移动，$Weight(.)$ 表示上下文权重。对于高分辨率网络，我们提出了 ACM 方法的两个实例，DCM 和 GCM，它们利用了并行多分辨率架构的优势。

2.2 密集上下文建模 (DCM)

我们引入 DCM 操作来密集建模一个阶段所有分辨率分支的特征的空间上下文关系。在第 n 阶段，将所有 n 个分支的输入特征汇聚到分辨率最低的 $H_n\times W_n$。然后，所有汇集的特性被连接在一起，这样就可以在并行的上下文特性上密集地执行上下文转移最后，将移位后的上下文特征上采样到相应的分辨率，并分布回相应的分支进行后续的上下文加权。这个实例化实现 ACM 如下:

$$\begin{cases} \hat{X}_k = ACPool(H_n, W_n)(X_k) \\ \tilde{X} = Shift(Cat([\hat{X}_1, ..., \hat{X}_n])) \\ Y_k = \begin{cases} Weight(X_k, Upsamp(\tilde{X}_k)), 1 \le k \le n-1 \\ Weight(X_k, \tilde{X}_k), \qquad k = n \end{cases} \end{cases}$$

其中 $Cat(.)$ 和 $Upsamp(.)$ 分别表示特征拼接和上采样。X_k 表示具有 k^{th} 最高分辨率的输入张量。$\hat{X}$ 表示来自 k^{th} 分支的集合张量。$\hat{X}_k$ 表示移位的张量，它被分配到 k^{th} 分支，即 $\hat{X}_k$。Y_k 表示对应的 k^{th} 输出张量。

2.3 动态内核聚合 (DKA)

为了使 SCS 模块在卷积核很小的情况下也能学习到丰富的上下文信息，我们引入了一种 DKA 操作，通过基于输入图像的核关注权动态聚合多个核，增强了卷积核的输入依赖性。

标准卷积核由 4 维权矩阵 w 定义，权矩阵 w 分别决定核大小和输入输出通道。我们没有将不同卷积的输出特征串联起来，而是在计算卷积结果之前将核权矩阵 w_i 聚合起来，针对不同的输入动态生成不同的卷积核。DKA 运算计算不同卷积核上的注意权值，然后将元素加权乘积应用于注意权值和核权值。我们如下定义 DKA 操作:

$$Y = W^T(X)X$$

$$W(X) = \sum_i^N a_i(X) w_i$$

其中 $a_i(X)$ 为 i^{th} 卷积核的注意权值，$W(X)$ 为 N 个卷积核的聚合权值矩阵。

输入依赖注意权重 $a(X)$ 由输入 X 计算如下:

$$a(X) = Sigmoid(FC(ReLU(FC(GAP(X)))))$$

其中 $GAP(.)$ 表示全局平均池，$FC(.)$ 表示全连接层。两个函数 $Sigmoid(.)$ 和 $ReLU(.)$ 在两个完全连接的层之后用于非线性激活。

由于 DKA 操作发生在计算卷积结果之前，因此聚合后的核对每个输入特征图只进行一次卷积操作，不会扩大网络宽度。

为了在每个分辨率上分别建模全球空间依赖性，我们在网络的每个分支上应用 GCM 操作。当自适应上下文池的输出大小为 1×1 时，它是 ACM 的一个实例化。

$$Y = Weight(X, Shift(ACPool(1,1)(X)))1 \leq k \leq n$$

GCM 操作在包含丰富上下文信息的全局面中捕获具有相同分辨率的所有特征的空间关系，而 DCM 操作在包含更多像素信息的中等面中捕获不同分辨率的所有特征的空间关系。

同时，这两种操作都增加了跨特性的信息交换，因此可以更好地替代 shuffle 块中的 1×1 卷积，而不是 Lite-HRNet [17] 中的信道加权操作。

3 不足和改进

轻量级模型的发展使得神经网络更加高效, 从而能够广泛地应用到各种场景任务中. 一方面, 轻量级神经网络有更小的体积和计算量, 降低了对设备存储能力和计算能力的需求, 既可以装配到传统家电中使其更加智能化, 也可以将深度学习系统应用在虚拟现实、增强现实、智能安防和智能可穿戴设备等新兴技术中; 另一方面, 轻量级神经网络具有更快的运行速度和更短的延时, 能够对任务进行实时处理, 对于在线学习、增量学习和分布式学习有重大意义; 另外, 实时处理的神经网络能够满足自动驾驶技术的需求, 提高自动驾驶的安全性. 轻量级神经网络将对于人工智能技术的普及、建立智能化城市起不可或缺的作用

目前的深度学习网络模型只能针对某一特定或同一类型的数据集，实现特殊的邻域任务。如何使用跨不同数据集的知识来加速优化过程, 是未来研究的热点。其他的挑战是联合优化深度神经网络流程的所有模型参数。到目前为止, 深度神经网络的通用自动化仍处于起步阶段, 许多问题尚未得到解决。然而, 这仍然是一个令人兴奋的领域, 并且未来的工作的方向需要强调其突出的实用性。

为了提高整体网络结构对于特征图信息的学习能力，而注意机制在深卷积神经网络 (DCNN) 已经成为流行推动特征图信息中远程依赖关系, 特定于像素级别的关注。可后续将目前泛化性较好的注意力机制模块加入进整体的网络模型中，在较小的损失速度的情况下，能够大幅度的提升精度。在整体网络结构的尾部采用 PP-PicoDet [18] 提出了 CSP-PAN 结构，使用 1×1 的卷积将特征的通道数与 BackBone 输出的最小通道数进行统一，从而减少计算量，并保证特征融合性能不受影响。

References

[1] A. Bulat and G. Tzimiropoulos, "Human pose estimation via convolutional part heatmap regression," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VII*, ser. Lecture Notes in Computer Science, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9911. Springer, 2016, pp. 717–732.

[2] J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik, "Human pose estimation with iterative error feedback," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 4733–4742.

[3] J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014, pp. 1799–1807.

[4] P. Hu and D. Ramanan, "Bottom-up and top-down reasoning with hierarchical rectified gaussians," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 5600–5609.

[5] J. Charles, T. Pfister, D. R. Magee, D. C. Hogg, and A. Zisserman, "Personalizing human video pose estimation," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 3063–3072.

[6] G. Gkioxari, A. Toshev, and N. Jaitly, "Chained predictions using convolutional neural networks," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, ser. Lecture Notes in Computer Science, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9908. Springer, 2016, pp. 728–743.

[7] Z. Cao, T. Simon, S. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 1302–1310.

[8] E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele, "Deepercut: A deeper, stronger, and faster multi-person pose estimation model," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI*, ser. Lecture Notes in Computer Science, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9910. Springer, 2016, pp. 34–50.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[11] M. Andriluka, L. Pishchulin, P. V. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis," in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 2014, pp. 3686–3693.

[12] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, ser. Lecture Notes in Computer Science, D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., vol. 8693. Springer, 2014, pp. 740–755.

[13] Y. Chen, Z. Wang, Y. Peng, Z. Zhang, G. Yu, and J. Sun, "Cascaded pyramid network for multi-person pose estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7103–7112.

[14] C. Yu, B. Xiao, C. Gao, L. Yuan, L. Zhang, N. Sang, and J. Wang, "Lite-hrnet: A lightweight high-resolution network," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 10 440–10 450.

[15] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, W. Liu, and B. Xiao, "Deep high-resolution representation learning for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 10, pp. 3349–3364, 2021.

[16] K. Sun, B. Xiao, D. Liu, and J. Wang, "Deep high-resolution representation learning for human pose estimation," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 5693–5703.

[17] C. Cui, T. Gao, S. Wei, Y. Du, R. Guo, S. Dong, B. Lu, Y. Zhou, X. Lv, Q. Liu, X. Hu, D. Yu, and Y. Ma, "Pp-lcnet: A lightweight CPU convolutional neural network," *CoRR*, vol. abs/2109.15099, 2021.

[18] G. Yu, Q. Chang, W. Lv, C. Xu, C. Cui, W. Ji, Q. Dang, K. Deng, G. Wang, Y. Du, B. Lai, Q. Liu, X. Hu, D. Yu, and Y. Ma, "Pp-picodet: A better real-time object detector on mobile devices," *CoRR*, vol. abs/2111.00902, 2021.

附录

DenseContextModeling

```
class DenseContextModeling(nn.Module):
    def __init__(self, channels, reduction):
        super().__init__()
        num_branches = len(channels)
        self.reduction = reduction[num_branches-2]
        self.channels = channels
        total_channel = sum(channels)
        mid_channels = total_channel // self.reduction
        ##这个是ACM中的adaptive context pooling, a context mask,一个1*1卷积和softmax部分
        self.conv_mask = nn.ModuleList([
        nn.Conv2d(channels[i], 1, kernel_size=1, stride=1, padding=0, bias=True)
        for i in range(len(channels))
                                        ])
        self.softmax = nn.Softmax(dim=2)

        ##这个是shift操作——2个1*1卷积和一个sigmoid函数
        self.channel_attention = nn.Sequential(
            nn.Conv2d(total_channel, mid_channels, kernel_size=1, stride=1, padding=0, bias=
                False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, total_channel, kernel_size=1, stride=1, padding=0, bias=
                True),
            nn.Sigmoid()
            )
##这个就是真正的ACM实现
    def global_spatial_pool(self, x, mini_size, i):
        batch, channel, height, width = x.size()
        mini_height, mini_width = mini_size
        # [N, C, H, W]
        x_m = x
        # [N, C, H * W]
        x_m = x_m.view(batch, channel, height * width)
        # [N, MH * MW, C, (H * W) / (MH * MW)]
        x_m = x_m.view(batch, mini_height * mini_width, channel, (height * width) // (
            mini_height * mini_width))
        # [N, 1, H, W]
        mask = self.conv_mask[i](x)
        # [N, 1, H * W]
        mask = mask.view(batch, 1, height * width)
        # [N, 1, H * W]
        mask = self.softmax(mask)
        # [N, MH * MW, (H * W) / (MH * MW)]
        mask = mask.view(batch, mini_height * mini_width, (height * width) // (mini_height *
            mini_width))
        # [N, MH * MW, (H * W) / (MH * MW), 1]
        mask = mask.unsqueeze(-1)
        # [N, MH * MW, C, 1]
        x = torch.matmul(x_m, mask)
        # [N, C, MH * MW, 1]
        x = x.showspacespermute(0, 2, 1, 3)
        # [N, C, MH, MW]
```

```
        x = x.view(batch, channel, mini_height, mini_width)
        return x
    def forward(self, x):
        mini_size = x[-1].size()[-2:]
        out = [self.global_spatial_pool(s, mini_size, i) for s, i in zip(x[:-1], range(len(x)))
            ] + [x[-1]]
        out = torch.cat(out, dim=1)
        out = self.channel_attention(out)
        out = torch.split(out, self.channels, dim=1)
        out = [s * F.interpolate(a, size=s.size()[-2:], mode='nearest') for s, a in zip(x, out)
            ]
        return out
```

GlobalContextModeling

```
class GlobalContextModeling(nn.Module):
    def __init__(self, channels, num_branch, reduction, with_cp=False):
        super().__init__()

        self.with_cp = with_cp

        self.reduction = reduction[num_branch]

        mid_channels = channels // self.reduction

        self.conv_mask = nn.Conv2d(channels, 1, kernel_size=1, stride=1, padding=0, bias=True)
        self.softmax = nn.Softmax(dim=2)

        self.channel_attention = nn.Sequential(
            nn.Conv2d(channels, mid_channels, kernel_size=1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, channels, kernel_size=1, stride=1, padding=0, bias=True),
            nn.Sigmoid()
            )

        self.bn = nn.BatchNorm2d(channels)

    def global_spatial_pool(self, x):
        batch, channel, height, width = x.size()

        # [N, C, H, W]
        x_m = x
        # [N, C, H * W]
        x_m = x_m.view(batch, channel, height * width)
        # [N, 1, C, H * W]
        x_m = x_m.unsqueeze(1)
        # [N, 1, H, W]
        mask = self.conv_mask(x)
        # [N, 1, H * W]
        mask = mask.view(batch, 1, height * width)
        # [N, 1, H * W]
        mask = self.softmax(mask)
        # [N, 1, H * W, 1]
        mask = mask.unsqueeze(-1)
        # [N, 1, C, 1]
```

```
        x = torch.matmul(x_m, mask)
        # [N, C, 1, 1]
        x = x.showspacespermute(0, 2, 1, 3)

        return x

    def forward(self, x):

        def _inner_forward(x):
            identity = x

            x = self.global_spatial_pool(x)
            x = self.channel_attention(x)
            x = self.bn(identity * x)

            return x

        if self.with_cp and x.requires_grad:
            x = cp.checkpoint(_inner_forward, x)
        else:
            x = _inner_forward(x)

            return x
```

DynamicSplitConvolution

```
class DynamicSplitConvolution(nn.Module):

    def __init__(self, channels, stride, num_branch, num_groups, num_kernels, with_cp=False):
        super().__init__()

        self.with_cp = with_cp

        self.num_groups = num_groups[num_branch]
        self.num_kernels = num_kernels[num_branch]

        self.split_channels = _split_channels(channels, self.num_groups)

        self.conv = nn.ModuleList([
        ConvBN(
        self.split_channels[i],
        self.split_channels[i],
        kernel_size=i * 2 + 3,
        stride=stride,
        padding=i + 1,
        groups=self.split_channels[i],
        num_kernels=self.num_kernels)
        for i in range(self.num_groups)
        ])

        def forward(self, x):

            def _inner_forward(x):
            if self.num_groups == 1:
            x = self.conv[0](x)
            else:
```

```
            x_split = torch.split(x, self.split_channels, dim=1)
            x = [conv(t) for conv, t in zip(self.conv, x_split)]
            x = torch.cat(x, dim=1)
            x = channel_shuffle(x, self.num_groups)

            return x

            if self.with_cp and x.requires_grad:
                x = cp.checkpoint(_inner_forward, x)
            else:
                x = _inner_forward(x)

            return x
```