On the Integration of Self-Attention and Convolution

Xuran Pan[1] Chunjiang Ge[1] Rui Lu[1] Shiji Song[1] Guanfu Chen[2] Zeyi Huang[2] Gao Huang[1,3*]
[1]Department of Automation, BNRist, Tsinghua University, Beijing, China
[2]Huawei Technologies Ltd., China
[3]Beijing Academy of Artificial Intelligence, Beijing, China
{pxr18, gecj20, r-lu21}@mails.tsinghua.edu.cn
{chenguanfu1, huangzeyi2}@huawei.com, {shijis, gaohuang}@tsinghua.edu.cn

Abstract

Convolution and self-attention are two powerful techniques for representation learning, and they are usually considered as two peer approaches that are distinct from each other. In this paper, we show that there exists a strong underlying relation between them, in the sense that the bulk of computations of these two paradigms are in fact done with the same operation. Specifically, we first show that a traditional convolution with kernel size $k \times k$ can be decomposed into k^2 individual 1×1 convolutions, followed by shift and summation operations. Then, we interpret the projections of queries, keys, and values in self-attention module as multiple 1×1 convolutions, followed by the computation of attention weights and aggregation of the values. Therefore, the first stage of both two modules comprises the similar operation. More importantly, the first stage contributes a dominant computation complexity (square of the channel size) comparing to the second stage. This observation naturally leads to an elegant integration of these two seemingly distinct paradigms, i.e., a ***mixed*** *model that enjoys the benefit of both self-**A**ttention and **C**onvolution (ACmix), while having minimum computational overhead compared to the pure convolution or self-attention counterpart. Extensive experiments show that our model achieves consistently improved results over competitive baselines on image recognition and downstream tasks. Code and pre-trained models will be released at* https://github.com/LeapLabTHU/ACmix *and* https://gitee.com/mindspore/models.

1. Introduction

Recent years have witnessed the vast development of convolution and self-attention in computer vision. Convolution neural networks (CNNs) are widely adopted on image recognition [20, 24], semantic segmentation [9] and object detection [39], and achieve state-of-the-art performances on various benchmarks. On the other hand, self-attention is first introduced in natural language processing [1, 43], and also shows great potential in the fields of image generation and super-resolution [10, 35]. More recently, with the advent of vision transformers [7, 16, 38], attention-based modules have achieved comparable or even better performances than their CNN counterparts on many vision tasks.

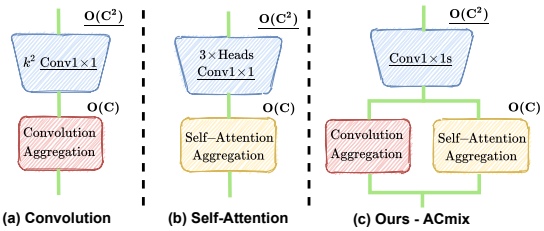


Figure 1. A sketch of ACmix. We explore a closer relationship between convolution and self-attention in the sense of sharing the same computation overhead (1×1 convolutions), and combining with the remaining lightweight aggregation operations. We show the computation complexity of each block *w.r.t* the feature channel.

Despite the great success that both approaches have achieved, convolution and self-attention modules usually follow different design paradigms. Traditional convolution leverages an aggregation function over a localized receptive field according to the convolution filter weights, which are shared in the whole feature map. The intrinsic characteristics impose crucial inductive biases for image processing. Comparably, the self-attention module applies a weighted average operation based on the context of input features, where the attention weights are computed dynamically via a similarity function between related pixel pairs. The flexibility enables the attention module to focus on different regions adaptively and capture more informative features.

Considering the different and complementary properties of convolution and self-attention, there exists a potential

*Corresponding author.

possibility to benefit from both paradigms by integrating these modules. Previous work has explored the combination of self-attention and convolution from several different perspectives. Researches from early stages, e.g., SENet [23], CBAM [47], show that self-attention mechanism can serve as an augmentation for convolution modules. More recently, self-attention modules are proposed as individual blocks to substitute traditional convolutions in CNN models, e.g., SAN [54], BoTNet [41]. Another line of research focuses on combining self-attention and convolution in a single block, e.g., AA-ResNet [3], Container [17], while the architecture is limited in designing independent paths for each module. Therefore, existing approaches still treat self-attention and convolution as distinct parts, and the underlying relations between them have not been fully exploited.

In this paper, we seek to unearth a closer relationship between self-attention and convolution. By decomposing the operations of these two modules, we show that they heavily rely on the same 1×1 convolution operations. Based on this observation, we develop a mixed model, named ACmix, and integrate self-attention and convolution elegantly with minimum computational overhead. Specifically, we first project the input feature maps with 1×1 convolutions and obtain a rich set of intermediate features. Then, the intermediate features are reused and aggregated following different paradigms, i.e, in self-attention and convolution manners respectively. In this way, ACmix enjoys the benefit of both modules, and effectively avoids conducting expensive projection operations twice.

To summarize, our contributions are two folds:

(1) A strong underlying relation between self-attention and convolution is revealed, providing new perspectives on understanding the connections between two modules and inspirations for designing new learning paradigms.

(2) An elegant integration of the self-attention and convolution module, which enjoys the benefits of both worlds, is presented. Empirical evidence demonstrates that the hybrid model outperforms its pure convolution or self-attention counterpart consistently.

2. Related Work

Convolution neural networks [27, 28], which use convolution kernels to extract local features, have become the most powerful and conventional technique for various vision tasks [20, 25, 40]. Meanwhile, self-attention also demonstrated its prevailing performance on a broad range of language tasks like BERT and GPT3 [4, 14, 37]. Theoretical analysis [11] indicates that, when equipped with sufficiently large capacity, self-attention can express the function class of any convolution layers. Therefore, a line of research recently explores the possibility of adopting the self-attention mechanism into vision tasks [16, 23]. There are two mainstream methods, one uses self-attention as building blocks in a network [7, 33, 55], and another views self-attention and convolution as complementary parts [6, 29, 45].

2.1. Self-Attention only

Inspired by the power of self-attention's expressive ability in long-range dependencies [14, 43], a march of work endeavours to solely use self-attention as elementary building blocks to construct the model for vision tasks [2, 7, 16]. Some works [38, 54] show that self-attention can become a stand-alone primitive for vision models which completely substitute convolutional operations. Recently, Vision Transformer [16] shows that given enough data, we can treat an image as a sequence of 256 tokens and leverage Transformer models [43] to achieve competitive results in image recognition. Furthermore, transformer paradigm is adopted in detection [2, 7, 57], segmentation [46, 53, 55], point cloud recognition [18, 33] and other vision tasks [8, 35].

2.2. Attention enhanced Convolution

Multiple previously proposed attention mechanisms over images suggest it can overcome the limitation of locality for convolutional networks. Therefore, many researchers explore the possibility of employing attention modules or utilizing more relational information to enhance the functionality of convolutional networks. Particularly, Squeeze-and-Excitation (SE) [23] and Gather-Excite (GE) [22] reweigh the map for each channel. BAM [34] and CBAM [47] independently reweigh both channels and spatial locations to better refine the feature map. AA-Resnet [3] augments certain convolutional layers by concatenating attention maps from another independent self-attention pipeline. BoTNet [41] substitutes convolutions with self-attention modules at late stages of the model. Some work aims at designing a more flexible feature extractor by aggregating information from a wider range of pixels. Hu *et al*. [21] proposed a local-relation approach to adaptively determine aggregation weights based on the compositional relationship of local pixels. Wang *et al*. proposed non-local network [45], which increases the receptive field by introducing non-local blocks that compare similarity among global pixels.

2.3. Convolution enhanced Attention

With the advent of Vision Transformer [16], numerous transformer-based variants have been proposed and achieved significant improvements on computer vision tasks. Among which exist researches focusing on complementing transformer models with convolution operations to introduce additional inductive biases. CvT [48] adopts convolution in the tokenization process and utilize stride convolution to reduce computation complexity of self-attention. ViT with convolutional stem [50] proposes to add convolutions at the early stage to achieve stabler training. CSwin Transformer [15] adopts a convolution-based positional en-

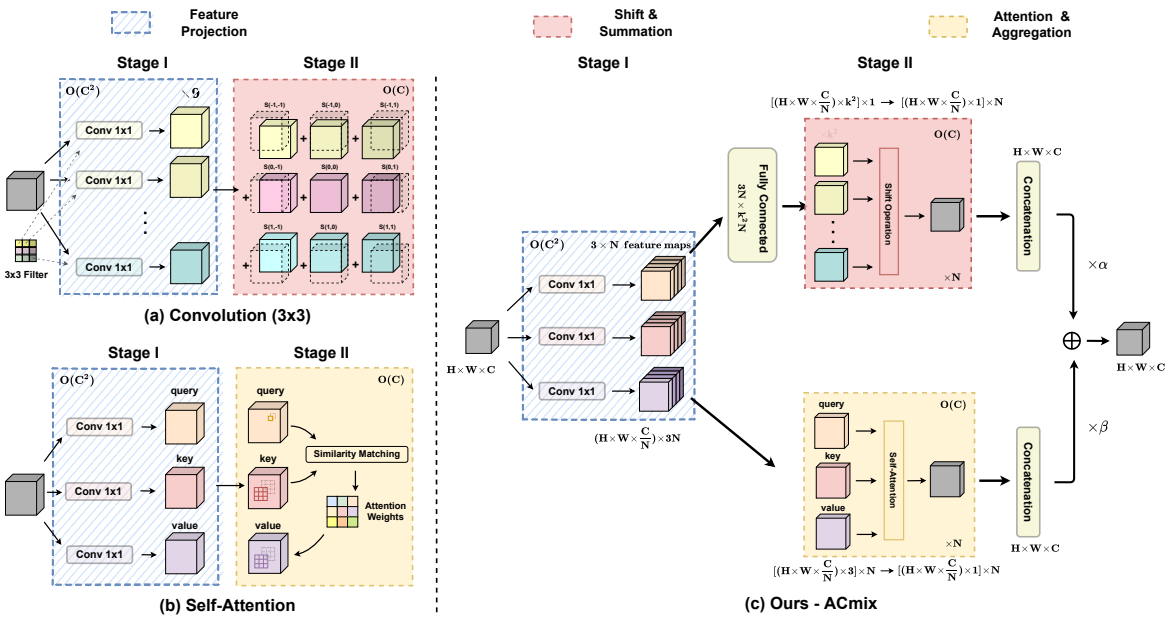


Figure 2. An illustration of the proposed hybrid module. The left figures show the pipeline of traditional convolution and self-attention modules. (a) Convolution. The output of a 3×3 convolution can be decomposed as a summation of shifted feature maps, where each feature map is obtained by performing a 1×1 convolution concerning the kernel weights from a certain position. $s(x, y)$ corresponds to the Shift operation defined in Sec.3.1. (b) Self-Attention. Input feature map is first projected as queries, keys, and values with 1×1 convolutions. The attention weights computed by queries and keys are adopted to aggregate the values. The right figure shows the pipeline of our module. (c) ACmix. At stage I, the input feature map is projected with three 1×1 convolutions. At stage II, the intermediate features are used following two paradigms respectively. Features from both paths are added together and serve as the final output. The computational complexity of each operation block is marked at the upper corner.

coding technique and shows improvements on downstream tasks. Conformer [36] combines Transformer with an independent CNN model to integrate both features.

3. Revisiting Convolution and Self-Attention

Convolution and self-attention have been widely known in their current forms. To better capture the relationship between these two modules, we revisit them from a novel view by decomposing their operations into separated stages.

3.1. Convolution

Convolution is one of the most essential parts of modern ConvNets. We first review the standard convolution operation and reformulate it from a different perspective. The illustration is shown in Fig.2(a). For simplicity, we assume the stride of convolution is 1.

Consider a standard convolution with the kernel $K \in \mathcal{R}^{C_{\text{out}}\times C_{\text{in}}\times k\times k}$, where k is the kernel size and $C_{\text{in}}, C_{\text{out}}$ are the input and output channel size. Given tensors $F \in \mathcal{R}^{C_{\text{in}}\times H\times W}, G \in \mathcal{R}^{C_{\text{out}}\times H\times W}$ as the input and output feature maps, where H, W denote the height and width, we denote $f_{ij} \in \mathcal{R}^{C_{\text{in}}}, g_{ij} \in \mathcal{R}^{C_{\text{out}}}$ as the feature tensors of pixel (i, j) corresponding to F and G respectively. Then, the standard convolution can be formulated as:

$$g_{ij} = \sum_{p,q} K_{p,q} f_{i+p-\lfloor k/2\rfloor, j+q-\lfloor k/2\rfloor}, \tag{1}$$

where $K_{p,q} \in \mathcal{R}^{C_{\text{out}}\times C_{\text{in}}}$, $p, q \in \{0, 1, \ldots, k-1\}$, represents the kernel weights with regard to the indices of the kernel position (p, q).

For convenience, we can rewrite Eq.(1) as the summation of the feature maps from different kernel positions:

$$g_{ij} = \sum_{p,q} g_{ij}^{(p,q)}, \tag{2}$$

with

$$g_{ij}^{(p,q)} = K_{p,q} f_{i+p-\lfloor k/2\rfloor, j+q-\lfloor k/2\rfloor}. \tag{3}$$

To further simplify the formulation, we define the **Shift** operation, $\tilde{f} \triangleq \text{Shift}(f, \Delta x, \Delta y)$, as:

$$\tilde{f}_{i,j} = f_{i+\Delta x, j+\Delta y}, \ \forall i, j, \tag{4}$$

where $\Delta x, \Delta y$ correspond to the horizontal and vertical displacements. Then, Eq.(3) can be rewritten as:

$$\begin{aligned} g_{ij}^{(p,q)} &= K_{p,q} f_{i+p-\lfloor k/2\rfloor, j+q-\lfloor k/2\rfloor} \\ &= \text{Shift}(K_{p,q} f_{ij}, p - \lfloor k/2\rfloor, q - \lfloor k/2\rfloor). \end{aligned} \tag{5}$$

As a result, the standard convolution can be summarized as

Module	Stg	Theoretical FLOPs(×hw)	Theoretical Params	ResNet 50 FLOPs(G)	ResNet 50 Params(M)
Conv	I	$k_c^2\,C^2$	$k_c^2\,C^2$	1.9(99%)	11.3(100%)
	II	$k_c^2\,C$	0	0.1(1%)	0(0%)
Self Attention	I	$3C^2$	$3C^2$	1.0(83%)	3.8(100%)
	II	$2k_a^2\,C$	0	0.2(17%)	0(0%)
ACmix	I	$3C^2$	$3C^2$	1.0(73%)	3.8(92%)
	II	$(k_c^2+2k_a^2)C$ $+(3k_c^2+k_c^4)C$	$3k_c^2N+k_c^4C$	0.4(27%)	0.3(8%)

Table 1. FLOPs and Parameters for different modules at two stages. $\mathbf{C}$: Input and output channel. $\mathbf{h}, \mathbf{w}$: Length and width of feature map. $\mathbf{k_c}$: Kernel size for convolution. $\mathbf{k_a}$: Kernel size for self-attention. $\mathbf{N}$: Head of self-attention. Numbers in red correspond to the additional FLOPs/Params introduced by ACmix. Percentages within the brackets are fractions of the whole module.

two stages:

$$\textbf{Stage I:}\quad \tilde{g}_{ij}^{(p,q)} = K_{p,q} f_{ij}, \tag{6}$$

$$\textbf{Stage II:}\quad g_{ij}^{(p,q)} = \mathrm{Shift}(\tilde{g}_{ij}^{(p,q)}, p-\lfloor k/2 \rfloor, q-\lfloor k/2 \rfloor), \tag{7}$$

$$g_{ij} = \sum_{p,q} g_{ij}^{(p,q)}. \tag{8}$$

At the first stage, the input feature map is linearly projected *w.r.t.* the kernel weights from a certain position, i.e., (p, q). This is the same as a standard 1×1 convolution. While in the second stage, the projected feature maps are shifted according to the kernel positions and finally aggregated together. It can be easily observed that most of the computational costs are performed in the 1×1 convolution, while the following shift and aggregation are lightweight.

3.2. Self-Attention

Attention mechanism has also been widely adopted in vision tasks. Comparing to the traditional convolution, attention allows the model to focus on important regions within a larger size context. We show the illustration in Fig.2(b).

Consider a standard self-attention module with N heads. Let $F \in \mathcal{R}^{C_{\text{in}} \times H \times W}, G \in \mathcal{R}^{C_{\text{out}} \times H \times W}$ denote the input and output feature. Let $f_{ij} \in \mathcal{R}^{C_{\text{in}}}, g_{ij} \in \mathcal{R}^{C_{\text{out}}}$ denote the corresponding tensor of pixel (i, j). Then, output of the attention module is computed as:

$$g_{ij} = \Big\Vert_{l=1}^{N} \left(\sum_{a,b \in \mathcal{N}_k(i,j)} \mathrm{A}(W_q^{(l)} f_{ij}, W_k^{(l)} f_{ab}) W_v^{(l)} f_{ab} \right), \tag{9}$$

where $\Vert$ is the concatenation of the outputs of N attention heads, and $W_q^{(l)}, W_k^{(l)}, W_v^{(l)}$ are the projection matrices for queries, keys and values. $\mathcal{N}_k(i, j)$ represents a local region of pixels with spatial extent k centered around (i, j), and $\mathrm{A}(W_q^{(l)} f_{ij}, W_k^{(l)} f_{ab})$ is the corresponding attention weight with regard to the features within $\mathcal{N}_k(i, j)$.

For the widely adopted self-attention modules in [21,38], the attention weights are computed as:

$$\mathrm{A}(W_q^{(l)} f_{ij}, W_k^{(l)} f_{ab}) = \underset{\mathcal{N}_k(i,j)}{\mathrm{softmax}} \left(\frac{(W_q^{(l)} f_{ij})^{\mathrm{T}} (W_k^{(l)} f_{ab})}{\sqrt{d}} \right), \tag{10}$$

where d is the feature dimension of $W_q^{(l)} f_{ij}$.

Also, multi-head self-attention can be decomposed into two stages, and reformulated as:

$$\textbf{Stage I:}\ q_{ij}^{(l)} = W_q^{(l)} f_{ij}, k_{ij}^{(l)} = W_k^{(l)} f_{ij}, v_{ij}^{(l)} = W_v^{(l)} f_{ij}, \tag{11}$$

$$\textbf{Stage II:}\ g_{ij} = \Big\Vert_{l=1}^{N} \left(\sum_{a,b \in \mathcal{N}_k(i,j)} \mathrm{A}(q_{ij}^{(l)}, k_{ab}^{(l)}) v_{ab}^{(l)} \right). \tag{12}$$

Similar to the traditional convolution in Sec.3.1, 1×1 convolutions are first conducted in stage I to project the input feature as query, key and value. On the other hand, Stage II comprises the calculation of the attention weights and aggregation of the value matrices, which refers to gathering local features. The corresponding computational cost is also proved to be minor comparing to Stage I, following the same pattern as convolution.

3.3. Computational Cost

To fully understand the computation bottleneck of the convolution and self-attention modules, we analyse the floating-point operations (FLOPs) and the number of parameters at each stage and summarize in Tab.1. It is shown that theoretical FLOPs and parameters at Stage I of convolution have quadratic complexity with regard to the channel size C, while the computational cost for Stage II is linear to C and no additional training parameters are required.

A similar trend is also found for the self-attention module, where all training parameters are preserved at Stage I. As for the theoretical FLOPs, we consider a normal case in a ResNet-like model where $k_a = 7$ and $C = 64, 128, 256, 512$ for various layer depths. It is explicitly shown that Stage I consumes a heavier operation as $3C^2 > 2k_a^2 C$, and the discrepancy is more distinct as channel size grows.

To further verify the validity of our analysis, we also summarize the actual computational costs of the convolution and self-attention modules in a ResNet50 model in Tab.1. We practically add up the costs of all 3×3 convolution (or self-attention) modules to reflect the tendency from the model perspective. It is shown that 99% computation of convolution and 83% of self-attention are conducted at Stage I, which are consistent with our theoretical analysis.

4. Method

4.1. Relating Self-Attention with Convolution

The decomposition of self-attention and convolution modules in Sec.3 has revealed deeper relations from various

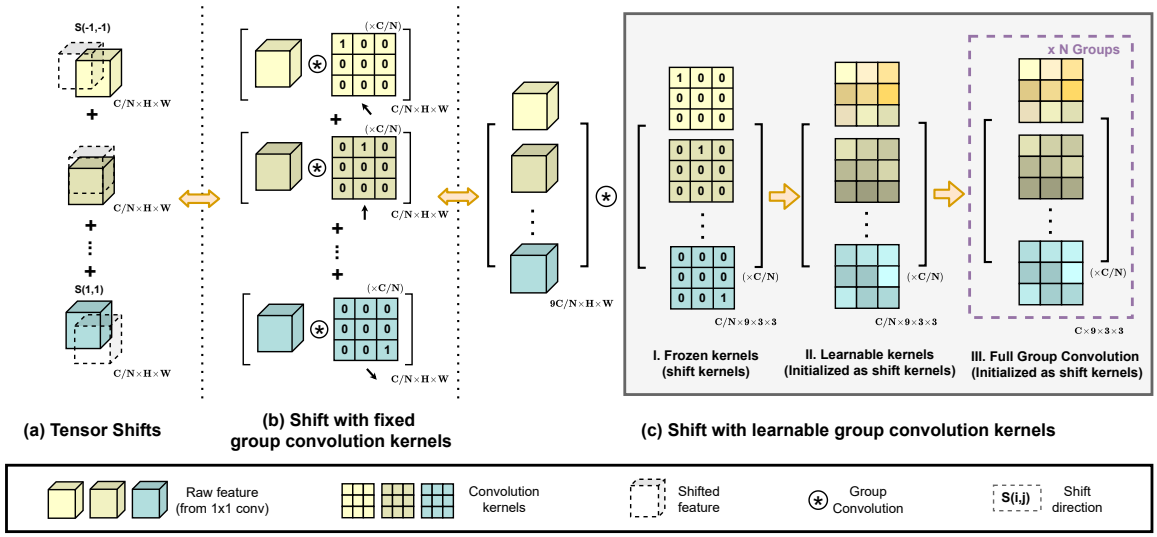


Figure 3. Practical improvements on shift operations. (a) Simple implementation with tensor shifts. (b) Fast implementation with carefully designed group convolution kernels. (c) Further adaptations with learnable kernels and multiple convolution groups.

perspectives. First, the two stages play quite similar roles. Stage I is a feature learning module, where both approaches share the same operations by performing 1×1 convolutions to project features into deeper spaces. On the other hand, stage II corresponds to the procedure of feature aggregation, despite the differences in their learning paradigms.

From the computation perspective, the 1×1 convolutions conducted at Stage I of both convolution and self-attention modules require a quadratic complexity of theoretical FLOPs and parameters with regard to the channel size C. Comparably, at stage II both modules are lightweight or nearly free of computation.

As a conclusion, the above analysis shows that (1) Convolution and self-attention practically share the same operation on projecting the input feature maps through 1×1 convolutions, which is also the computation overhead for both modules. (2) Although crucial for capturing semantic features, the aggregation operations at stage II are lightweight and do not acquire additional learning parameters.

4.2. Integration of Self-Attention and Convolution

The aforementioned observations naturally lead to an elegant integration of convolution and self-attention. As both modules share the same 1×1 convolution operations, we can only perform the projection once, and reuse these intermediate feature maps for different aggregation operations respectively. The illustration of our proposed mixed module, ACmix, is shown in Fig.2(c).

Specifically, ACmix also comprises two stages. At Stage I, input feature is projected by three 1×1 convolutions and reshaped into N pieces, respectively. Thus, we obtain a rich set of intermediate features containing $3\times N$ feature maps.

At Stage II, they are used following different paradigms. For the self-attention path, we gather the intermediate features into N groups, where each group contains three pieces of features, one from each 1×1 convolution. The corresponding three feature maps serve as queries, keys, and values, following the traditional multi-head self-attention modules (Eq.(12)). For the convolution path with kernel size k, we adopt a light fully connected layer and generate k^2 feature maps. Consequently, by shifting and aggregating the generated features (Eq.(7),(8)), we process the input feature in a convolution manner, and gather information from a local receptive field like the traditional ones.

Finally, outputs from both paths are added together and the strengths are controlled by two learnable scalars:

$$F_{\text{out}} = \alpha F_{\text{att}} + \beta F_{\text{conv}}. \tag{13}$$

4.3. Improved Shift and Summation

As shown in Sec.4.2 and Fig.2, intermediate features in the convolution path follow the shift and summation operations as conducted in traditional convolution modules. Despite that they are theoretically lightweight, shifting tensors towards various directions practically breaks the data locality and is difficult to achieve vectorized implementation. This may greatly impair the actual efficiency of our module at the inference time.

As a remedy, we resort to applying **depthwise convolution with fixed kernels** as a replacement of the inefficient tensor shifts, as shown in Fig.3 (b). Take $\text{Shift}(f, -1, -1)$ as an example, shifted feature is computed as:

$$\tilde{f}_{c,i,j} = f_{c,i-1,j-1},\ \forall c, i, j, \tag{14}$$

where c represents each channel of the input feature.

On the other hand, if we denote convolution kernel (ker-

nel size $k=3$) as:

$$K_c = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \forall c, \tag{15}$$

the corresponding output can be formulated as:

$$f_{c,i,j}^{(\text{dwc})} = \sum_{p,q\in\{0,1,2\}} K_{c,p,q} f_{c,i+p-\lfloor k/2\rfloor, j+q-\lfloor k/2\rfloor} \tag{16}$$

$$= f_{c,i-1,j-1} = \tilde{f}_{c,i,j}, \quad \forall c,i,j. \tag{17}$$

Therefore, with carefully designed kernel weights for specific shift directions, the convolution outputs are equivalent to the simple tensor shifts (Eq.(14)). To further incorporate with the summation of features from different directions, we concatenate all the input features and convolution kernels respectively, and formulate shift operation as a single group convolution, as depicted in Fig.3 (c.I). This modification enables our module with higher computation efficiency.

On this basis, we additionally introduce several adaptations to enhance the flexibility of the module. As shown in Fig.3 (c.II), we release the convolution kernel as learnable weights, with shift kernels as initialization. This improves the model capacity while maintaining the ability of original shift operations. We also use multiple groups of convolution kernels to match the output channel dimension of convolution and self-attention paths, as depicted in Fig.3 (c.III).

4.4. Computational Cost of ACmix

For better comparison, we summarize the FLOPs and parameters of ACmix in Tab.1. The computational cost and training parameters at Stage I are the same as self-attention and lighter than traditional convolution (e.g., 3×3 conv). At Stage II, ACmix introduces additional computation overhead with a light fully connected layer and a group convolution described in Sec.4.3, whose computation complexity is linear with regard to channel size C and comparably minor with Stage I. The practical cost in a ResNet50 model shows similar trends with theoretical analysis.

4.5. Generalization to Other Attention Modes

With the development of the self-attention mechanism, numerous researches have focused on exploring variations of the attention operator to further promote the model performance. Patchwise attention proposed by [54] incorporates information from all features in the local region as the attention weights to replace the original softmax operation. Window attention adopted by Swin-Transformer [32] keeps the same receptive field for tokens in the same local window to save computational cost and achieve fast inference speed. ViT and DeiT [16,42], on the other hand, consider global attention to retaining long-range dependencies within a single layer. These modifications are proved to be effective under specific model architectures.

Under the circumstance, it is worth noticing that our proposed ACmix is independent of self-attention formulations, and can be readily adopted on the aforementioned variants. Specifically, the attention weights can be summarized as:

$$\textbf{(Patchwise)}\ \mathrm{A}(q_{ij},k_{ab}) = \phi([q_{ij},[k_{ab}]_{a,b\in\mathcal{N}_k(i,j)}]), \tag{18}$$

$$\textbf{(Window)}\ \mathrm{A}(q_{ij},k_{ab}) = \underset{a,b\in\mathcal{W}_k(i,j)}{\mathrm{softmax}}\left(q_{ij}^{\mathrm{T}}k_{ab}/\sqrt{d}\right), \tag{19}$$

$$\textbf{(Global)}\ \mathrm{A}(q_{ij},k_{ab}) = \underset{a,b\in\mathcal{W}}{\mathrm{softmax}}\left(q_{ij}^{\mathrm{T}}k_{ab}/\sqrt{d}\right), \tag{20}$$

where $[\cdot]$ refers to feature concatenation, $\phi(\cdot)$ represents two linear projection layers with an intermediate nonlinear activation, $\mathcal{W}_k(i,j)$ is the specialized receptive field for each query token, and $\mathcal{W}$ represents the whole feature map (Please refer to the original paper for further details). Then, the computed attention weights can be applied to Eq.(12) and fits into the general formulation.

5. Experiments

In this section, we empirically validate ACmix on ImageNet classification, semantic segmentation, and object detection tasks, and compare with state-of-the-art models. See Appendix for detailed dataset and training configurations.

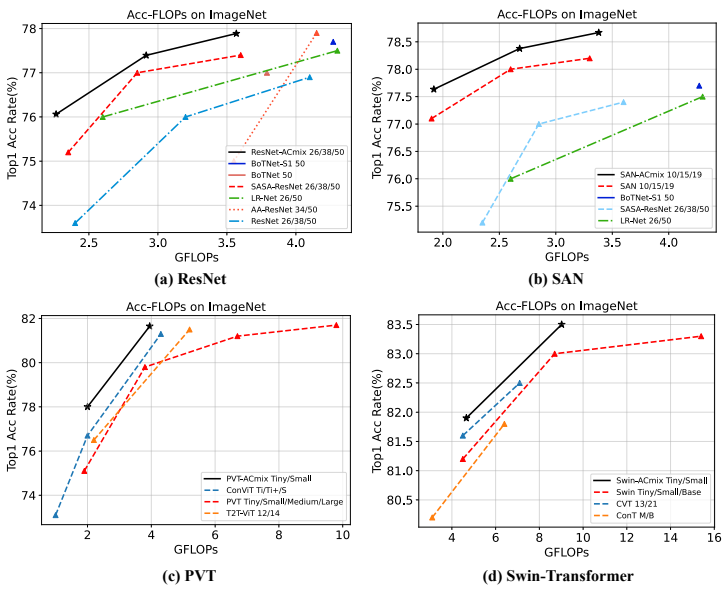
5.1. ImageNet Classification

Implementation. We practically implement ACmix on 4 baseline models, including ResNet [20], SAN [54], PVT [44] and Swin-Transformer [32]. We also compare our models with competitive baselines, i.e., SASA [38], LR-Net [21], AA-ResNet [3], BoTNet [41], T2T-ViT [52], ConViT [12], CVT [48], ConT [51] and Conformer [36].

Results. We show the classification results in Fig.4. For ResNet-ACmix models, our model outperforms all baselines with comparable FLOPs or parameters. For example, ResNet-ACmix 26 achieves same top-1 accuracy as SASA-ResNet 50 with 80% FLOPs. With similar FLOPs, our model outperforms SASA by $0.35\%-0.8\%$. The superiority against other baselines is even larger. For SAN-ACmix, PVT-ACmix and Swin-ACmix, our models achieve consistent improvements. As a showcase, SAN-ACmix 15 outperforms SAN 19 with 80% FLOPs. PVT-ACmix-T shows comparable performance with PVT-Large, with only 40% FLOPs. Swin-ACmix-S achieves higher accuracy than Swin-B with 60% FLOPs.

5.2. Downstream Tasks

Semantic Segmentation We evaluate the effectiveness of our models on a challenging scene parsing dataset, ADE20K [56], and display the results on two segmentation approaches, Semantic-FPN [26] and UperNet [49].



Method	Params	Flops	Top-1
ResNet 26	13.7M	2.4G	73.6
ResNet-ACmix 26	10.6M	2.3G	**76.1** (+2.5)
ResNet 38	19.6M	3.2G	76.0
ResNet-ACmix 38	14.6M	2.9G	**77.4** (+1.4)
ResNet 50	25.6M	4.1G	76.9
ResNet-ACmix 50	18.6M	3.6G	**77.8** (+0.9)
SAN 10	11.8M	1.9G	77.1
SAN-ACmix 10	12.1M	1.9G	**77.6** (+0.5)
SAN 15	16.2M	2.6G	78.0
SAN-ACmix 15	16.6M	2.7G	**78.4** (+0.4)
SAN 19	20.5M	3.3G	78.2
SAN-ACmix 19	21.2M	3.4G	**78.7** (+0.5)
PVT-T	13M	1.9G	75.1
PVT-ACmix-T	13M	2.0G	**78.0** (+2.9)
PVT-S	25M	3.8G	79.8
PVT-ACmix-S	25M	3.9G	**81.7** (+1.9)
Swin-T	29M	4.5G	81.3
Swin-ACmix-T	30M	4.6G	**81.9** (+0.6)
Swin-S	50M	8.7G	83.0
Swin-ACmix-S	51M	9.0G	**83.5** (+0.5)

Figure 4. Comparisons of FLOPS and parameters against accuracy on ImageNet classification task. Methods in (a) adapts from ResNet-50 with tradition attentions, methods in (b) adapts from SAN with patchwise attentions, methods in (c) adapts from PVT with global attentions, and methods in (d) adapts from Swin-Transformer with window attentions.

Backbones are pretrained on ImageNet-1K. It is shown that ACmix achieves improvements under all settings.

Object Detection We also conduct experiments on the COCO benchmark [31]. Tab.3 and Tab.4 display the result of ResNet-based models and Transformer-based models with various detection heads, including RetinaNet [30], Mask R-CNN [19] and Cascade Mask R-CNN [5]. We can observe that ACmix consistently outperform baselines with similar parameters or FLOPs. This further validate the effectiveness of ACmix when transfered to downstream tasks.

5.3. Practical Inference Speed

We further investigate the practical inference speed of our method under an Ascend 910 environment with MindSpore, a deep learning computing framework for mobile, edge, and cloud scenarios. We summarize the results in Tab.5. Comparing to PVT-S, our model achieves 1.3x fps with comparable mAP. When it comes to the larger model, the superiority is more distinct. ACmix outperforms PVT-L 1.9mAP with 1.8x fps.

5.4. Ablation Study

To evaluate the effectiveness of different components in ACmix, we conduct a series of ablation studies.

Combining the output of both paths. We explore how different combinations of the convolution and self-attention outputs influence the model performances. We conduct experiments with multiple combination methods and summarize the results in Tab.6. We also show the performances of models adopting only one path, Swin-T for self-attention, and Conv-Swin-T for convolution by replacing the window attention with traditional 3×3 convolutions. As we can observe, the combination of convolution and self-attention modules consistently outperforms models with a single path. Fixing the ratio of convolution and self-attention for all operators also leads to worse performance. Comparably, using learned parameters imposes higher flexibility for ACmix, and the strength for convolution and self-attention paths can be adaptively adjusted according to the position of the filter in the whole network.

Group Convolution Kernels. We also conduct ablations on the choices of group convolution kernels, as we have

Method	Backbone	Schd	Params	Flops	val mIoU
Semantic FPN	PVT-T	40k	17M	158G	37.1
	ACmix	40k	17M	160G	**42.7** (+5.6)
	PVT-S	40k	28M	225G	42.4
	ACmix	40k	29M	228G	**46.4** (+4.0)
UperNet	Swin-T	160k	60M	945G	44.5
	ACmix	160k	60M	950G	**45.3** (+0.8)
	Swin-S	160k	81M	1038G	47.6
	ACmix	160k	81M	1043G	**48.7** (+1.1)

Table 2. ADE20K segmentation with Transformer-based models.

Method	Backbone	Schd	Flops	mAP	mAP_{50}	mAP_{75}
RetinaNet	ResNet 50	1x	250G	36.7	56.0	39.0
	SASA	1x	226G	36.8	54.6	39.3
	ACmix	1x	230G	**38.3**	**56.2**	**40.0**
RetinaNet	SAN 19	1x	229G	38.2	56.0	41.1
	ACmix	1x	233G	**39.1**	**58.9**	**41.6**

Table 3. COCO Object detection with ResNet-based models.

Method	Backbone	Schd	Flops	mAP	mAP_{50}	mAP_{75}
RetinaNet	PVT-T	1x	230G	36.7	56.9	38.9
	ACmix	1x	232G	**40.5**	**61.2**	**42.7**
RetinaNet	PVT-T	3x	230G	39.4	59.8	42.0
	ACmix	3x	232G	**42.0**	**62.8**	**44.6**
Mask R-CNN	Swin-T	3x	272G	46.0	67.8	50.4
	ACmix	3x	275G	**47.0**	**69.0**	**51.8**
Cascade Mask R-CNN	Swin-T	3x	750G	50.5	69.3	54.9
	ACmix	3x	754G	**51.1**	**69.8**	**55.6**

Table 4. COCO Object detection with Transformer-based models.

Method	**Backbone**	**mAP**	**FPS**
RetinaNet	PVT-S	**42.2**	50.3
	ACmix	**42.0**	**67.7** **(x1.3)**
RetinaNet	PVT-L	43.4	24.3
	ACmix	**45.3**	**43.9** **(x1.8)**

Table 5. Practical inference speed on COCO. FPS is test on a single Ascend 910 with input image size (3, 576, 576).

Method	α	β	**Params**	**Flops**	**top-1**
Swin-T	1	-	29M	4.5G	81.3
Conv-Swin-T	-	1	39M	4.5G	80.5
Swin-ACmix-T	1	1	30M	4.6G	81.5
	α	1	30M	4.6G	81.6
	α	1-α	30M	4.6G	81.5
	α	β	30M	4.6G	**81.9**

Table 6. Ablation study on combining methods of two paths. The final output is computed as $F_{\text{out}} = \alpha \cdot F_{\text{att}} + \beta \cdot F_{\text{conv}}$.

Shift Module	**Flops**	**Top-1**	**FPS**
Tensor Shifts (Fig.3a)	4.6G	81.4	313
Fixed Kernel (Fig.3c.I)	4.7G	81.4	419
Random Init (Fig.3c.II w/o init)	4.7G	81.5	419
Group Conv (Full)	4.7G	**81.9**	**419**

Table 7. Ablation study of shift modules implementations based on Swin-Transformer-T. FPS is test on a single RTX2080Ti GPU with maximum batchsize.

shown in Sec.4.3 and Fig.3. We empirically show the effectiveness of each adaptation, and its influence on practical inference speed in Tab.7. By substituting the tensor shifts with group convolutions, inference speed is greatly boosted. Also, using learnable convolution kernels and carefully-designed initialization enhance model flexibility and contribute to the final performance.

5.5. Bias towards Different Paths.

It is also valuable to see that ACmix introduces two learnable scalars α, β to combine the outputs from both paths (Eq.14). This leads to a by-product of our module, where α and β practically reflect the model's bias towards convolution or self-attention at different depths.

We conduct parallel experiments and show the learned

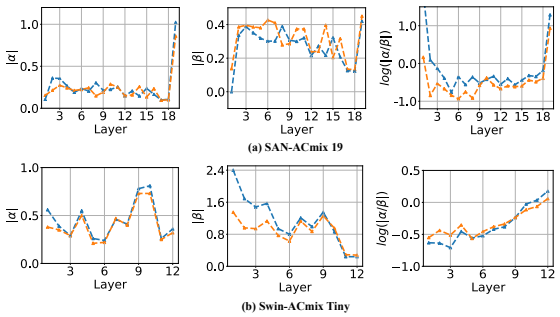


Figure 5. $|\alpha|$, $|\beta|$ and $log(|\alpha/\beta|)$ from different layers of SAN-ACmix and Swin-ACmix. Lines in the same plot correspond to parallel experiments. The final output is computed as $F_{\text{out}} = \alpha \cdot F_{\text{att}} + \beta \cdot F_{\text{conv}}$.

parameters α, β from different layers of SAN-ACmix, and Swin-ACmix models in Fig.5. The left and middle plots show the changing tendency of rates for self-attention and convolution paths respectively. The variation of the rates in different experiments is relatively small, especially when layers go deeper. This observation shows a stable preference for deep models towards the different design patterns. A more distinct trend is shown in the right plot, where the ratio between two paths is explicitly presented. We can see that convolution can serve as good feature extractors at the early stages of the Transformer models. At the middle stage of the network, the model tends to leverage the mixture of both paths with an increasing bias towards convolution. At the last stage, self-attention shows superiority over convolution. This is also consistent with the design patterns in the previous works where self-attention is mostly adopted in the last stages to replace the original 3×3 convolution [3, 41], and convolutions at early stages are proved to be more effective for vision transformers [50].

6. Conclusion

In this paper, we explore a close relationship between two powerful techniques, convolution and self-attention. By decomposing the operations of both modules, we show that they share the same computation overhead on projecting the input feature maps. On this basis, we take a step forward and propose a hybrid operator to integrate self-attention and convolution modules by sharing the same heavy operations. Extensive results on image classification and object detection benchmarks demonstrate the effectiveness and efficiency of the proposed operator.

Acknowledgements

This work is supported in part by the National Science and Technology Major Project of the Ministry of Science and Technology of China under Grants 2018AAA0100701, the National Natural Science Foundation of China under Grants 61906106 and 62022048, and Huawei Technologies Ltd.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Josh Beal, Eric Kim, E. Tzeng, Dong Huk Park, Andrew Zhai, and Dmitry Kislyuk. Toward transformer-based object detection. *ArXiv*, abs/2012.09958, 2020.

[3] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le. Attention augmented convolutional networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3286–3295, 2019.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[5] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018.

[6] Yue Cao, J. Xu, Stephen Lin, Fangyun Wei, and H. Hu. Gcnet: Non-local networks meet squeeze-excitation networks and beyond. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 1971–1980, 2019.

[7] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

[8] Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. Pre-trained image processing transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12299–12310, June 2021.

[9] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[10] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[11] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. In *International Conference on Learning Representations*, 2020.

[12] Stéphane d'Ascoli, Hugo Touvron, Matthew Leavitt, Ari Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. *arXiv preprint arXiv:2103.10697*, 2021.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[15] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. *arXiv preprint arXiv:2107.00652*, 2021.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[17] Peng Gao, Jiasen Lu, Hongsheng Li, Roozbeh Mottaghi, and Aniruddha Kembhavi. Container: Context aggregation network. *arXiv preprint arXiv:2106.01401*, 2021.

[18] Menghao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, R. Martin, and S. Hu. Pct: Point cloud transformer. *ArXiv*, abs/2012.09688, 2020.

[19] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3464–3473, 2019.

[22] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Andrea Vedaldi. Gather-excite: Exploiting feature context in convolutional neural networks. 2018.

[23] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[24] Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens Van Der Maaten, and Kilian Weinberger. Convolutional networks with dense connectivity. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[25] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[26] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6399–6408, 2019.

[27] A. Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90, 2012.

[28] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[29] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 510–519, 2019.

[30] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[31] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014.

[32] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[33] Xuran Pan, Zhuofan Xia, Shiji Song, Li Erran Li, and Gao Huang. 3d object detection with pointformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7463–7472, June 2021.

[34] Jongchan Park, Sanghyun Woo, Joon-Young Lee, and In So Kweon. Bam: Bottleneck attention module. *arXiv preprint arXiv:1807.06514*, 2018.

[35] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR, 2018.

[36] Zhiliang Peng, Wei Huang, Shanzhi Gu, Lingxi Xie, Yaowei Wang, Jianbin Jiao, and Qixiang Ye. Conformer: Local features coupling global representations for visual recognition. *arXiv preprint arXiv:2105.03889*, 2021.

[37] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[38] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[39] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015.

[40] K. Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[41] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16519–16529, 2021.

[42] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[44] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021.

[45] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[46] Yuqing Wang, Zhaoliang Xu, Xinlong Wang, Chunhua Shen, Baoshan Cheng, Hao Shen, and Huaxia Xia. End-to-end video instance segmentation with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8741–8750, 2021.

[47] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[48] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. *arXiv preprint arXiv:2103.15808*, 2021.

[49] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 418–434, 2018.

[50] Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross Girshick. Early convolutions help transformers see better. *arXiv preprint arXiv:2106.14881*, 2021.

[51] Haotian Yan, Zhe Li, Weijian Li, Changhu Wang, Ming Wu, and Chuang Zhang. Contnet: Why not use convolution and transformer at the same time? *arXiv preprint arXiv:2104.13497*, 2021.

[52] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zihang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. *arXiv preprint arXiv:2101.11986*, 2021.

[53] Dong Zhang, Hanwang Zhang, J. Tang, Meng Wang, Xiansheng Hua, and Qianru Sun. Feature pyramid transformer. *ArXiv*, abs/2007.09451, 2020.

[54] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. In *Proceedings of*

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10076–10085, 2020.

[55] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6881–6890, 2021.

[56] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 633–641, 2017.

[57] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. In *International Conference on Learning Representations*, 2021.

Appendix

A. Model Architectures

We summarize the architectures of ResNet 26/38/50 [20], SAN 10/15/19 [54], PVT-T/S [44], Swin-T/S [32], and their respective ACmix version in Tab 9∼12. For fair comparison, we only substitute the original 3×3 convolution or self-attention module with our proposed operator in the modified models.

B. Dataset and Training Setup

ImageNet. ImageNet 2012 [13] comprises 1.28 million training images and 50,000 validation images from 1000 different classes. For ResNet-based models, we follow the training schedule in [54] and train all the models for 100 epochs. We use SGD with batchsize 256 on 8 GPUs. Cosine learning rate is adopted with the base learning rate set to 0.1. We apply standard data augmentation, including random cropping, random horizontal flipping and normalization. We use label smoothing with coefficient 0.1. For experiments on Transformer-based models, including PVT and Swin-Transformer, we follow training configurations in the original paper.

COCO. COCO dataset [31] is a standard object detection benchmark and we use a subset of 80k samples as training set and 35k for validation. For ResNet and SAN models, we train the network by SGD and 8 GPU are used with a batchsize of 16. For PVT and Swin-Transformer models, we train the network by adamw. Backbone networks are respectively pretrained on ImageNet dataset following the same training configurations in the original paper. We follow the "1x" learning schedule to train the whole network for 12 epochs and divide the learning rate by 10 at the 8th and 11th epoch respectively. For several transformer-based models, we follow the configurations in the original paper, and additionally experiment "3x" schedule with 36 epochs. We apply standard data augmentation, that is resize, random flip and normalize. Learning rate is set at 0.01 and linear warmup is used in the first 500 iterations. We follow the "1x" learning schedule training the whole network for 12 epochs and divide the learning rate by 10 at the 8th and 11th epoch respectively. For several transformer-based models, we follow the configurations in the original paper, and test with "3x" schedule. All mAP results in the main paper are tested with input image size (3, 1333, 800).

ADE20K. ADE20K [56] is a widely-used semantic segmentation dataset, containing 150 categories. ADE20K has 25K images, with 20K for training, 2K for validation, and another 3K for testing. For two baseline models, PVT and Swin-Transformer, we follow the training configurations in their original paper respectively. For PVT, we implement the backbone models on the basis of Semantic FPN [26]. We optimize the models using AdamW with an initial learning rate of 1e-4 for 80k iterations. For Swin-Transformer, we implement the backbone models on the basis of UperNet [49]. We use the AdamW optimizer with an initial learning rate of 6e-5 and a linear warmup of 1,500 iterations. Models are trained for a total of 160K iterations. We randomly resize and crop the image to 512 × 512 for training, and rescale to have a shorter side of 512 pixels during testing.

C. Hyper-parameters

For ResNet-ACmix models, we set $N = 4$ for all the experiments.

For SAN-ACmix models, the channel dimension for queries, keys and values are different in the original model [54]. Given input features with channel dimension C, queries and keys are projected to features with $C/4$ channels, while values are projected to features with C channels. Therefore, when implementing our ACmix operator, we divide values into 4 groups, where the divided groups have the same channel dimension $C/4$. The following self-attention and convolution operations follow the same *patchwise* attention in [54] and the same designing pipeline as we stated in Sec.4, respectively.

For PVT-ACmix and Swin-ACmix models, we follow the configurations in the original model [32].

$k_a = 7$ and $k_c = 3$ is set for all experiments, unless stated otherwise.

D. Positional Encoding

Positional encoding is widely adopted in self-attention modules, while not used in SAN and PVT models. Therefore, we follow this setting and only adopt positional encoding in the ResNet-ACmix models and Swin-ACmix. Specifically, the popular relative positional encoding [38] is adopted when computing the attention weights:

$$\mathrm{A}(q_{ij}, k_{ab}) = \underset{\mathcal{N}(i,j)}{\mathrm{softmax}} \left((q_{ij}^{\mathrm{T}} k_{ab} + B_{ij,ab}) / \sqrt{d} \right), \quad (21)$$

where q, k, B represent queries, keys and relative positional encodings respectively. We didn't include positional encoding in the analysis for computation complexity in the Tab.1 of the main paper, as the *patchwise* attention proposed in [54] demonstrate the effectiveness of self-attention modules without adopting it. Nevertheless, the computation cost for positional encoding is also linear with respect to the channel dimension C, which is also comparably minor to the feature projection operations. Therefore, considering the positional encoding doesn't affect our main statement.

E. Practical Costs for Other Models

We also summarize the practical FLOPs and Parameters for convolution, self-attention and ACmix based on various models introduced in the Experiment section. The numbers are shown in Tab.8. Similar to ResNet 50, more than 60% of the computation are performed at Stage I of the self-attention module in SAN and Swin models. Meanwhile, it also demonstrates that ACmix only introduces minimum computational cost to integrate both convolution and self-attention modules based on various model structures.

Module	**Stage**	**ResNet 50 GFLOPs**	**SAN 19 GFLOPs**	**Swin-T GFLOPs**
Convolution	I	$1.85\ (99\%)$	-	-
	II	$0.01\ (1\%)$	-	-
Self-Attention	I	$0.96\ (83\%)$	$1.29\ (64\%)$	$1.04\ (68\%)$
	II	$0.19\ (17\%)$	$0.72\ (36\%)$	$0.49\ (32\%)$
ACmix	I	$0.96\ (73\%)$	$1.29\ (60\%)$	$1.04\ (62\%)$
	II	$0.35\ (27\%)$	$0.89(40\%)$	$0.64\ (38\%)$

Table 8. Practical FLOPs and Parameters for different modules based on various models. Numbers within the brackets are their fractions of the whole module. SAN 19 and Swin-T models are designed with all self-attention modules, thus not applicable for the traditional convolution module.

stage	output	ResNet-26 (**ACmix**)	ResNet-38 (**ACmix**)	ResNet-50 (**ACmix**)
res1	112×112	7×7 conv, 64, stride 2	7×7 conv, 64, stride 2	7×7 conv, 64, stride 2
res2	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 1 \times 1 \text{ conv}, 64 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 64 \\ 1 \times 1 \text{ conv}, 256 \end{bmatrix} \times 1$	3×3 max pool, stride 2 $\begin{bmatrix} 1 \times 1 \text{ conv}, 64 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 64 \\ 1 \times 1 \text{ conv}, 256 \end{bmatrix} \times 2$	3×3 max pool, stride 2 $\begin{bmatrix} 1 \times 1 \text{ conv}, 64 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 64 \\ 1 \times 1 \text{ conv}, 256 \end{bmatrix} \times 3$
res3	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv}, 128 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 128 \\ 1 \times 1 \text{ conv}, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1 \text{ conv}, 128 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 128 \\ 1 \times 1 \text{ conv}, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1 \text{ conv}, 128 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 128 \\ 1 \times 1 \text{ conv}, 512 \end{bmatrix} \times 4$
res4	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv}, 256 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 256 \\ 1 \times 1 \text{ conv}, 1024 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1 \text{ conv}, 256 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 256 \\ 1 \times 1 \text{ conv}, 1024 \end{bmatrix} \times 5$	$\begin{bmatrix} 1 \times 1 \text{ conv}, 256 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 256 \\ 1 \times 1 \text{ conv}, 1024 \end{bmatrix} \times 6$
res5	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv}, 512 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 512 \\ 1 \times 1 \text{ conv}, 2048 \end{bmatrix} \times 1$	$\begin{bmatrix} 1 \times 1 \text{ conv}, 512 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 512 \\ 1 \times 1 \text{ conv}, 2048 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1 \text{ conv}, 512 \\ 3 \times 3 \text{ conv } (\textbf{ACmix}), 512 \\ 1 \times 1 \text{ conv}, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax

Table 9. Architectures of ResNet-based models with and without ACmix modules.

layers	output	SAN-10 (**ACmix**)	SAN-15 (**ACmix**)	SAN-19 (**ACmix**)
Input	224×224	1×1 conv, 64	1×1 conv, 64	1×1 conv, 64
Transition	112×112	2×2 max pool, stride 2 1×1 conv, 64	2×2 max pool, stride 2 1×1 conv, 64	2×2 max pool, stride 2 1×1 conv, 64
Block	112×112	$\begin{bmatrix} 3 \times 3 \text{ sa } (\textbf{ACmix}), 16 \\ 1 \times 1 \text{ conv}, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3 \text{ sa } (\textbf{ACmix}), 16 \\ 1 \times 1 \text{ conv}, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3 \text{ sa } (\textbf{ACmix}), 16 \\ 1 \times 1 \text{ conv}, 64 \end{bmatrix} \times 3$
Transition	56×56	2×2 max pool, stride 2 1×1 conv, 256	2×2 max pool, stride 2 1×1 conv, 256	2×2 max pool, stride 2 1×1 conv, 256
Block	56×56	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 64 \\ 1 \times 1 \text{ conv}, 256 \end{bmatrix} \times 1$	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 64 \\ 1 \times 1 \text{ conv}, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 64 \\ 1 \times 1 \text{ conv}, 256 \end{bmatrix} \times 3$
Transition	28×28	2×2 max pool, stride 2 1×1 conv, 512	2×2 max pool, stride 2 1×1 conv, 512	2×2 max pool, stride 2 1×1 conv, 512
Block	28×28	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 128 \\ 1 \times 1 \text{ conv}, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 128 \\ 1 \times 1 \text{ conv}, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 128 \\ 1 \times 1 \text{ conv}, 512 \end{bmatrix} \times 4$
Transition	14×14	2×2 max pool, stride 2 1×1 conv, 1024	2×2 max pool, stride 2 1×1 conv, 1024	2×2 max pool, stride 2 1×1 conv, 1024
Block	14×14	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 256 \\ 1 \times 1 \text{ conv}, 1024 \end{bmatrix} \times 4$	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 256 \\ 1 \times 1 \text{ conv}, 1024 \end{bmatrix} \times 5$	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 256 \\ 1 \times 1 \text{ conv}, 1024 \end{bmatrix} \times 6$
Transition	7×7	2×2 max pool, stride 2 1×1 conv, 2048	2×2 max pool, stride 2 1×1 conv, 2048	2×2 max pool, stride 2 1×1 conv, 2048
Block	7×7	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 512 \\ 1 \times 1 \text{ conv}, 2048 \end{bmatrix} \times 1$	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 512 \\ 1 \times 1 \text{ conv}, 2048 \end{bmatrix} \times 2$	$\begin{bmatrix} 7 \times 7 \text{ sa } (\textbf{ACmix}), 512 \\ 1 \times 1 \text{ conv}, 2048 \end{bmatrix} \times 3$
Classification	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax

Table 10. Architectures of SAN-based models with and without ACmix modules.

stage	output	layer name	PVT-T (**ACmix**)	PVT-S (**ACmix**)
res1	56×56	Patch Embedding	$P_1 = 4;\ C_1 = 64$	$P_1 = 4;\ C_1 = 64$
		Transformer	$\begin{bmatrix} R_1 = 8 \\ N_1 = 1 \\ E_1 = 8 \end{bmatrix}$ (**ACmix**) $\times 2$	$\begin{bmatrix} R_1 = 8 \\ N_1 = 1 \\ E_1 = 8 \end{bmatrix}$ (**ACmix**) $\times 3$
res2	28×28	Patch Embedding	$P_2 = 2;\ C_2 = 128$	$P_2 = 2;\ C_2 = 128$
		Transformer	$\begin{bmatrix} R_2 = 4 \\ N_2 = 2 \\ E_2 = 8 \end{bmatrix}$ (**ACmix**) $\times 2$	$\begin{bmatrix} R_2 = 4 \\ N_2 = 2 \\ E_2 = 8 \end{bmatrix}$ (**ACmix**) $\times 4$
res3	14×14	Patch Embedding	$P_3 = 2;\ C_3 = 320$	$P_3 = 2;\ C_3 = 320$
		Transformer	$\begin{bmatrix} R_3 = 2 \\ N_3 = 5 \\ E_3 = 4 \end{bmatrix}$ (**ACmix**) $\times 2$	$\begin{bmatrix} R_3 = 2 \\ N_3 = 5 \\ E_3 = 4 \end{bmatrix}$ (**ACmix**) $\times 6$
res4	7×7	Patch Embedding	$P_4 = 2;\ C_4 = 512$	$P_4 = 2;\ C_4 = 512$
		Transformer	$\begin{bmatrix} R_4 = 1 \\ N_4 = 8 \\ E_4 = 4 \end{bmatrix}$ (**ACmix**) $\times 2$	$\begin{bmatrix} R_4 = 1 \\ N_4 = 8 \\ E_4 = 4 \end{bmatrix}$ (**ACmix**) $\times 3$

Table 11. Architectures of PVT-based models with and without ACmix modules.

stage	output	Swin-T (**ACmix**)	Swin-S (**ACmix**)
res1	56×56	concat 4×4, 96, LN	concat 4×4, 96, LN
		$\begin{bmatrix} 7 \times 7 \text{ window} \\ \text{dim } 96, \text{head } 3 \end{bmatrix}$ (**ACmix**) $\times 2$	$\begin{bmatrix} 7 \times 7 \text{ window} \\ \text{dim } 96, \text{head } 3 \end{bmatrix}$ (**ACmix**) $\times 2$
res2	28×28	concat 2×2, 192, LN	concat 4×4, 192, LN
		$\begin{bmatrix} 7 \times 7 \text{ window} \\ \text{dim } 192, \text{head } 6 \end{bmatrix}$ (**ACmix**) $\times 2$	$\begin{bmatrix} 7 \times 7 \text{ window} \\ \text{dim } 192, \text{head } 6 \end{bmatrix}$ (**ACmix**) $\times 2$
res3	14×14	concat 2×2, 384, LN	concat 4×4, 384, LN
		$\begin{bmatrix} 7 \times 7 \text{ window} \\ \text{dim } 384, \text{head } 12 \end{bmatrix}$ (**ACmix**) $\times 6$	$\begin{bmatrix} 7 \times 7 \text{ window} \\ \text{dim } 384, \text{head } 12 \end{bmatrix}$ (**ACmix**) $\times 18$
res4	7×7	concat 2×2, 768, LN	concat 4×4, 768, LN
		$\begin{bmatrix} 7 \times 7 \text{ window} \\ \text{dim } 768, \text{head } 24 \end{bmatrix}$ (**ACmix**) $\times 2$	$\begin{bmatrix} 7 \times 7 \text{ window} \\ \text{dim } 768, \text{head } 24 \end{bmatrix}$ (**ACmix**) $\times 2$

Table 12. Architectures of Swin-based models with and without ACmix modules.