

ELSA: Enhanced Local Self-Attention for Vision Transformer

Jingkai Zhou^{12*} Pichao Wang^{2†} Fan Wang² Qiong Liu^{1‡} Hao Li² Rong Jin²

¹South China University of Technology ²Alibaba Group

201510105876@mail.scut.edu.cn, liuqiong@scut.edu.cn

{pichao.wang, fan.w, lihao.lh, jinrong.jr}@alibaba-inc.com

Abstract

Self-attention is powerful in modeling long-range dependencies, but it is weak in local finer-level feature learning. The performance of local self-attention (LSA) is just on par with convolution and inferior to dynamic filters, which puzzles researchers on whether to use LSA or its counterparts, which one is better, and what makes LSA mediocre. To clarify these, we comprehensively investigate LSA and its counterparts from two sides: channel setting and spatial processing. We find that the devil lies in the generation and application of spatial attention, where relative position embeddings and the neighboring filter application are key factors. Based on these findings, we propose the enhanced local self-attention (ELSA) with Hadamard attention and the ghost head. Hadamard attention introduces the Hadamard product to efficiently generate attention in the neighboring case, while maintaining the high-order mapping. The ghost head combines attention maps with static matrices to increase channel capacity. Experiments demonstrate the effectiveness of ELSA. Without architecture / hyperparameter modification, drop-in replacing LSA with ELSA boosts Swin Transformer [48] by up to +1.4 on top-1 accuracy. ELSA also consistently benefits VOLO [83] from D1 to D5, where ELSA-VOLO-D5 achieves 87.2 on the ImageNet-1K without extra training images. In addition, we evaluate ELSA in downstream tasks. ELSA significantly improves the baseline by up to +1.9 box Ap / +1.3 mask Ap on the COCO, and by up to +1.9 mIoU on the ADE20K. Code is available at <https://github.com/damo-cv/ELSA>.

1. Introduction

From upstream to downstream visual tasks, vision transformers [3, 22, 35, 59, 62, 89] have set off a revolution by achieving promising results. Behind the success, the multi-head self-attention (MHSA) plays a critical role, which generates attention maps to dynamically aggregate spatial in-

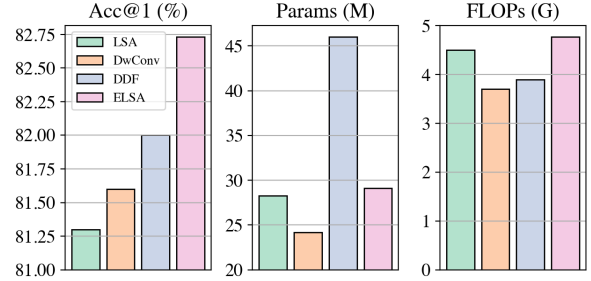


Figure 1. **Comparison of different layers on the Swin-T [48] architecture.** Local self-attention (LSA) in Swin-T is less efficient than depth-wise convolution (DwConv). Its top-1 accuracy is also inferior to dynamic filters like DDF [92]. Our ELSA surpasses these counterparts with a large margin while using the similar number of parameters and FLOPs as LSA.

formation, leading to greater flexibility and larger capacity. Recent studies [55, 82] demonstrated that MHSA tends to focus on local information in the first few layers of vision transformers. Several methods [11, 17, 83, 87] introduce inductive bias to force earlier layers to embed local details, which boosts the generalization ability of vision transformers. As a representative of them, Swin Transformer [48] brings locality to MHSA and makes great progress on a wide range of visual tasks.

However, one strange phenomenon appears in Swin Transformer. One can achieve similar performance when replacing local self-attention (LSA) in Swin Transformer with depth-wise convolution (DwConv) [13, 37] or dynamic filters. As shown in Figure 1, we replace the LSA in Swin-T [48] with DwConv and the decoupled dynamic filter (DDF) [92]. LSA only achieves similar top-1 accuracy as DwConv, which is lower than DDF, but it requires more floating-point operations (FLOPs). This phenomenon has also been observed in recent papers [17, 25, 32, 83], but they lack detailed analysis of the reason behind such performances, and it motivates us to raise a question: *what makes local self-attention mediocre?*

To answer this question, we thoroughly review LSA, DwConv, and dynamic filters from two key aspects: *chan-*

*Work done during an internship at Alibaba Group.

†Corresponding author, project lead

‡Corresponding author.

nel setting and spatial processing.

Channel setting. One straightforward difference between DwConv and LSA is the channel setting. DwConv applies different filters to different channels. LSA adopts the multi-head strategy, which shares filters (a.k.a attention maps) within each group of channels. In this work, we consider the channel setting of DwConv as a special case of the multi-head strategy, where the number of heads is set to the number of channels. One guess is that more heads in DwConv might be a critical factor in why it performs comparably to LSA. However, our experiments show that even if we set the head number of DwConv to be the same as that of LSA, DwConv still achieves similar or better accuracy. We also find that directly increasing the head number of LSA will not improve its performance. A new channel strategy is demanded by LSA to further improve accuracy.

Spatial processing. How to obtain and apply filters (or attention maps) to gather spatial information is another difference between DwConv, dynamic filters, and LSA. DwConv shares static filters across all feature pixels in a sliding window way. Dynamic filters [39, 42, 66, 83, 92] employ a bypath network, normally a 1×1 convolution, to generate spatial-specific filters, and apply these filters to the neighboring area of each pixel. LSA generates attention maps, which are also a kind of spatial-specific filters, via the dot product of the query and key matrices. LSA applies these attention maps to local windows. In this work, we unify the above three kinds of spatial processing into one paradigm, and fairly investigate them from parameterization, normalization, and filter application. We find that relative position embedding and neighboring filter application are two key factors that affect performance. However, calculating the dot product of queries and keys in the neighboring case is not computational-friendly. An efficient way of filter generation in the neighboring case is needed to replace the dot product while maintaining performance.

Based on these findings, we propose the enhanced local self-attention module (ELSA) to better embed local information. ELSA is composed of Hadamard attention and the ghost head module. In Hadamard attention, we replace the the dot product with the Hadamard product which is more computational-friendly in the neighboring case while maintaining the high-order mapping. The ghost head combines attention with static ghost head matrices, which effectively and efficiently improves channel capacity and performance. We empirically validate the performance of ELSA by drop-in replacing LSA / Outlooker [83] in Swin Transformer [48] / VOLO [83]. Without changing the architecture / hyperparameter of other parts, ELSA considerably improves the performance of Swin Transformer and VOLO, while introducing little overhead. In addition, we also demonstrate the superior performance of ELSA in downstream object detec-

tion and semantic segmentation tasks.

In short, we make the following contributions in this work:

- Extensively investigate the LSA and its counterparts so as to empirically reveal *what factors make LSA mediocre*.
- Propose the enhanced local self-attention (ELSA) to better embed local details by introducing Hadamard attention and the ghost head.
- Validate ELSA in both upstream and downstream tasks. The use of ELSA in drop-in replacement boosts baseline methods consistently.

2. Related Work

Vision Transformers. Transformer [65] is first proposed in the NLP task and achieves dominant performance [1, 20]. Recently, the pioneering work ViT [22] successfully applies the pure transformer-based architecture to computer vision, revealing the potential of transformer in handling visual tasks. Lots of follow-up studies are proposed [4, 5, 9, 12, 18, 21, 23, 24, 27–29, 31, 38, 41, 43, 45, 50, 52, 56, 76, 77, 80, 81, 84]. Many of them analyze the ViT [15, 17, 26, 32, 44, 55, 69, 73, 75, 82] and improve it via introducing locality to earlier layers [11, 17, 48, 64, 79, 83, 87]. In particular, Raghu *et al.* [55] observe that the first few layers in ViTs focus on local information. Li *et al.* [82] also demonstrate that the first few layers embed local details. Xiao *et al.* [75] and Wang *et al.* [68] find that introducing inductive bias, like convolution stem, can stabilize the training and improve the peak performance of ViTs. Similarly, Dai *et al.* [17] marry convolution with ViTs, improving the model generalization ability. Swin Transformer [48], as a milestone, also leverages local self-attention (LSA) to embed detailed information in high-resolution finer-level features. Despite these successes, several studies [17, 25, 32, 83] observe that the performance of LSA is just on par with convolution in both upstream and downstream tasks [32]. The reasons behind this phenomenon are not clear, and in-depth comparisons under the same conditions are valuable.

Dynamic filters. Convolution and depth-wise convolution [13] has been widely used in CNNs [34, 36, 37, 57, 60], while their content-agnostic nature limits the model flexibility and capacity. To solve this problem, dynamic filters are proposed one after another. One kind of dynamic filter [10, 51, 78, 88] predicts coefficients to combine several expert filters which are then shared across all spatial pixels. Another kind of dynamic filter [7, 39, 42, 58, 66, 67, 83, 92] generates spatial-specific filters. Specifically, the dynamic filter networks [39] use the separate network branches to predict a complete filter at each pixel. PAC [58] uses a fixed Gaussian kernel on adapting features to modify the standard convolution filter at each pixel. DRConv [7] extends CondConv [78] to each pixel. CARAFE [66] and

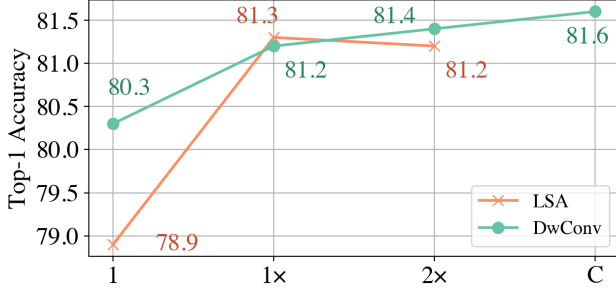


Figure 2. **Investigation of different channel settings.** Only the number of heads is changed each time for fair comparisons. The LSA version runs out of memory under setting C.

CARAFE++ [67] are the dynamic layers for upsampling and downsampling, where a channel-shared 2D filter is predicted at each pixel. Similarly, Involution [42] applies the CARAFE-like structure to feature extraction. VOLO [83] introduces the Outlooker to embed local details. DDF [92] decouples dynamic filters to spatial and channel ones, reducing computational overhead while achieving promising results. In this work, we observe that dynamic filters, like DDF [92], perform superior to LSA. Based on comparison and discussion, we empirically reveal the factors leading to such a phenomenon, and propose enhanced local self-attention (ELSA) to better embed local details.

3. Channel Setting

To figure out *what makes LSA mediocre*, we first focus on one of the most obvious differences between DwConv and LSA, *i.e.*, the channel setting. DwConv applies different filters to each channel. Differently, several dynamic filters [42, 66, 83] and LSA employ the multi-head strategy, which splits channels into multiple groups and shares the same filter within each group. In this work, we consider the setting of DwConv as a special case of the multi-head strategy, where the number of heads is equal to the number of channels. Therefore, comparing channel settings between LSA and DwConv is essentially investigating the performance of different head numbers.

Figure 2 shows the investigation results on the ImageNet-1K dataset. We compare two versions of Swin-T [48] under the same hyperparameters, and only modify the head setting each time. Setting 1 means that the number of heads is set to 1 for all layers. Setting 1x represents the original setting of Swin-T, that is, the head numbers of four stages are set to $\{3, 6, 12, 24\}$. Setting 2x means to double the original setting, *i.e.*, $\{6, 12, 24, 48\}$. Setting C denotes that the number of heads is set to the number of channels for all layers.

There has been a guess that more heads in the DwConv version increase channel capacity, thus may lead to compa-

table performance. However, we find that under the same channel setting, like 1x and 2x, the DwConv version still achieves similar performance as the LSA one. The DwConv version even exceeds the LSA one under setting 1, which demonstrates that the channel setting is not the essential factor leading to the strange phenomenon.

In addition, we observe that setting more heads than 1x does not benefit the LSA version. One possible reason is that more heads lead to fewer channels for each head generation. Thus, directly increasing the number of heads cannot improve channel capacity or performance. A new strategy is desired to further increase channel capacity and accuracy.

4. Spatial Processing

As the channel setting is not the critical factor, we seek the answer from the perspective of spatial processing. DwConv, dynamic filters, and LSA adopt different strategies to gather spatial information. We first review these strategies and unify them into one paradigm. Then, we fairly compare these strategies from three aspects.

4.1. Formulation

Conv / DwConv. Conv and DwConv do not generate filters. They share the static convolutional filters by sliding window. The spatial processing of them can be written as

$$\mathbf{f}_i = \sum_{j \in \Theta} \mathbf{w}_{j-i} \mathbf{x}_j \quad (1)$$

where \mathbf{f}_i represents the output at pixel i , \mathbf{x}_j represents the input at pixel j , \mathbf{w}_{j-i} is the filter weight, $j-i$ is the relative offset between pixel j and i . Take the filter size 3 as an example, $j-i$ corresponds to $\{(-1, -1), (-1, 0), (-1, 1), \dots, (1, 1)\}$. Θ notes the neighboring area around the pixel i .

Dynamic filters. Dynamic filters [42, 66, 83, 92] generate spatial-specific filters at each pixel via a bypath network. Taking Involution [42] as a representative, it uses a 1×1 convolution to generate filters at each pixel and applies them to the neighboring area of that pixel. It can be written as

$$\mathbf{f}_i = \sum_{j \in \Theta} \text{Norm}(\mathbf{x}_i \mathbf{w}) \mathbf{x}_j \quad (2)$$

where \mathbf{w} is the 1×1 convolutional weight of the filter generation branch. Norm represents the normalization method applied to the generated filters, which is the identity mapping in Involution. For other cases, such as DDF [92] and Outlooker [83], the normalization methods are the filter normalization [92] and softmax, respectively.

LSA. Unlike dynamic filters, LSA uses attention maps of local windows as spatial-specific filters. In terms of formulation, LSA can be written as

$$\mathbf{f}_i = \sum_{j \in \Omega} \text{Softmax}_j(\mathbf{q}_i \mathbf{k}_j) \mathbf{v}_j \quad (3)$$

where $\mathbf{q}_i, \mathbf{k}_j, \mathbf{v}_j$ are the query / key / value vectors at pixel i and j , respectively. They are generated from the input feature via linear mappings. Ω notes the local window area. Here, we omit the multi-head strategy for simplicity.

Unified paradigm. To unify the above strategies, we first consider \mathbf{w} in Equation 2 as a kind of relative position embedding, and consider \mathbf{w}_{j-i} in Equation 1 as a kind of relative position bias. Then, these spatial processing strategies can be unified into one paradigm, which can be written as

$$\mathbf{f}_i = \sum_{j \in \Phi} \text{Norm}_j(\mathbf{q}_i \mathbf{k}_j + \mathbf{q}_i \mathbf{r}_{j-i}^k + \mathbf{r}_{j-i}^q \mathbf{k}_j + \mathbf{r}_{j-i}^b \mathbf{v}_j) \quad (4)$$

where Φ can be either the local window Ω or the neighboring area Θ ; Norm_j can be either the identity mapping, the filter normalization, or softmax; \mathbf{r}_{j-i}^k and \mathbf{r}_{j-i}^q are relative position embeddings, and \mathbf{r}_{j-i}^b denotes the relative position bias.

DwConv, dynamic filters, and LSA are all special cases of this unified paradigm. For example, when only using \mathbf{r}_{j-i}^b as parameterization, leveraging the identity mapping as Norm_j , and adopting the neighboring area Θ as Φ , Equation 4 will degenerate to DwConv. Similarly, if the parameterization is set to $\mathbf{q}_i \mathbf{r}_{j-i}^k$, Norm_j is set to the identity mapping, and Φ is set to the neighboring area Θ , then Equation 4 becomes a variant of Involution, where \mathbf{r}_{j-i}^k is equivalent to \mathbf{w} in Equation 2. We can also get LSA from this paradigm by changing the parameterization, Norm_j , and Φ . Therefore, the spatial processing of LSA and its counterparts are essentially different in three factors: *parameterization*, *normalization*, and *filter application*. We then investigate each factor through extensive comparisons.

4.2. Investigation

Parameterization. We first compare the effect of different parameterizations by setting Φ as the local window and setting Norm_j as softmax. The results are exhibited in Table 1. As can be seen, when applying filters to local windows, the parameterization strategy of dynamic filters (Net2) is better than that of the standard LSA (Net1). Also, a variant of dynamic filters (Net6) is on par with the LSA in Swin-T. Moreover, Net7 indicates that combining the parameterization of LSA with dynamic filters can further boost the performance, when applying filters to local windows.

Normalization. Normalization is another factor that influences spatial processing. DwConv and Involution [42] adopt identity mapping as normalization. DDF [92] introduces the filter normalization. LSA and Outlooker [83] employ the softmax function as normalization. We fairly compare these options under the same conditions. We choose the Net7 in Table 1 as the baseline, and only change the normalization part each time. As can be seen in Table 2, the identity mapping causes the training crash, and softmax is

Model	$\mathbf{q}_i \mathbf{k}_j$	$\mathbf{q}_i \mathbf{r}_{j-i}^k$	$\mathbf{r}_{j-i}^q \mathbf{k}_j$	\mathbf{r}_{j-i}^b	Acc@1
Swin-T [†] [48]	✓			✓	81.3
Net1 [†] [48]	✓				80.1 _{-1.2}
Net2		✓			80.9 _{-0.4}
Net3			✓		80.9 _{-0.4}
Net4				✓	79.8 _{-1.5}
Net5		✓	✓		81.1 _{-0.2}
Net6		✓	✓	✓	81.3 _{0.0}
Net7	✓	✓	✓	✓	81.8 _{+0.5}

Table 1. **Comparison of different parameterization strategies.** Swin-T [48] is chosen as the baseline. Acc@1 means the top-1 accuracy. [†] denotes the results are reported in [48].

Model	Identity	Filter Norm	Softmax	Acc@1
	✓			crash
Net7		✓		81.4
			✓	81.8

Table 2. **Comparison of different normalization strategies.** We only switch the normalization of the Net7 in for fair comparison.

Model	Local Window	Neighboring	Acc@1
Swin-T [†] [48]	✓		81.3
		✓	81.4
Net6	✓		81.3
		✓	82.0
Net7	✓		81.8
		✓	82.0*
		✓	82.4 [‡]

Table 3. **Comparison of different filter applications.** We only switch the setting of the Φ in Equation 4 for fair comparison. [†] denotes the results are reported in [48]. * denotes that the model is implemented using SDC [87] and trained on one node. [‡] denotes that the model is implemented using the unfold operation and trained on two nodes.

better than the filter normalization. This indicates that the normalization part should not be blamed for the mediocre performance of LSA. For more comparisons, please refer to the appendix.

Filter application. The last factor of spatial processing is how filters are applied. LSA in Swin Transformer applies attention maps to non-overlapping local windows. In contrast, DwConv and dynamic filters apply filters to sliding neighboring areas. This difference can be described as using a different setting of Φ in Equation 4. We choose Swin-T, Net6, and Net7 as the baseline. Following the control variable principle, we only switch Φ in the first three stages of those models. We implement the neighboring case Net7

(noted as Net7-N) in two ways. We use SDC [87] to implement it for saving GPU memory, so that we can use one node (8 GPUs with 32G GPU memory) to train Net7-N like others. We also implemented the unfold version, but which consumes huge GPU memories and requires multiple nodes to train. Table 3 shows the comparison results. When applying filters to neighboring areas, both Net6 and Net7 get significantly improved, indicating that the neighboring application is critical for spatial processing.

4.3. Discussion

Key factors. Based on the above investigations, the factors that make LSA mediocre can be summarized in two folds: one key factor affecting performance is relative position embeddings. Even when applying filters to local windows, variants (Net5 and Net6) with relative position embedding \mathbf{r}_{j-i}^k and \mathbf{r}_{j-i}^q achieve similar performance as Swin-T. Net7 further surpasses Swin-T by 0.5% on top-1 accuracy. The other critical factor is the filter application. Applying filters on query-centered neighboring areas considerably boosts the performance of Net6 and Net7. So far, we can empirically answer the question we raised at beginning. The reason why DwConv can match the performance of LSA is because of the neighboring filter application. Without that, DwConv degenerates to a variant of Net4, which is inferior to LSA. Similarly, the reason why dynamic filters perform better than LSA is because of the relative position embedding and the neighboring filter application. Integrating these factors into LSA (Net7-N) achieves the best performance among all these variants.

Local window v.s. Neighboring. The peak performance of the local window version is worse than the neighboring one. Another disadvantage of the local window is that it requires strategies like window shifting to exchange information between windows, which limits the model design to have pairs of layers at each stage.

There is no such thing as a free lunch. The drawback of the neighboring version is low throughput. It is not easy to calculate the dot product between queries and keys in sliding neighboring areas. It requires sliding chunk [?], unfold operations, or specialized CUDA implementations, which are either memory-consuming or time-consuming. How to avoid the dot product while maintaining good performance is a challenging problem.

5. Enhanced Local Self-Attention

In addition to answering the question we raised, more importantly, we design a new local self-attention module that surpasses both the LSA in Swin Transformer and dynamic filters. This is accomplished with our enhanced local self-attention (ELSA), where the key techniques are Hadamard attention and the ghost head module. In terms

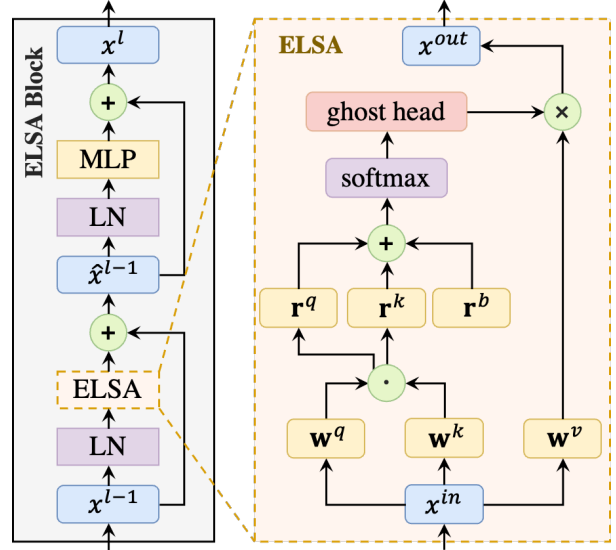


Figure 3. **Illustration of the ELSA block.** ELSA can seamlessly replace LSA or dynamic filters in models.

of formulation, ELSA can be written as

$$\mathbf{f}_i = \sum_{j \in \Theta} G(\mathbf{h}_{j-i}) \mathbf{v}_j \quad (5)$$

where \mathbf{h}_{j-i} is the Hadamard attention value, $G(\cdot)$ notes the ghost head mapping, \mathbf{f}_i is the output feature at pixel i , and \mathbf{v}_j is the value vector at pixel j . Figure 3 illustrates the overall structure of ELSA.

Hadamard attention. First, we review Net7-N and Net6-N, which get the highest and the second-highest accuracy in our investigation. As discussed above, Net7-N brings in difficulties on implementation and inference. In the neighboring case, $\mathbf{q}_i \mathbf{k}_j$ in Net7-N needs to be implemented using unfold, sliding chunks [87], or CUDA operations, which are either memory-consuming or time-consuming. As shown in Equation 6, Net6-N removes the dot product term, thus getting rid of difficulties in filter / attention generation.

$$\mathbf{f}_i = \sum_{j \in \Theta} \text{Softmax}_j(\mathbf{q}_i \mathbf{r}_{j-i}^k + \mathbf{r}_{j-i}^q \mathbf{k}_j + \mathbf{r}_{j-i}^b) \mathbf{v}_j \quad (6)$$

However, the performance of Net6-N is slightly worse than Net7-N. We find that the lower accuracy of Net6-N may be due to the lower mapping order. Specifically, Net-7 can be considered as a third-order mapping of the input \mathbf{x} , because Net-7 contains the second-order term $\mathbf{q}_i \mathbf{k}_j$ and combine it with the value \mathbf{v} . However, since Net6-N removes the dot product $\mathbf{q}_i \mathbf{k}_j$, which becomes a second-order mapping of the input \mathbf{x} .

One common hypothesis in deep learning is that the mappings with the higher order have stronger fitting abil-

ity [46]. Thus, we want to design a module that maintains the third-order mapping just like Net7-N, but without using the dot product of the query and key matrices. To accomplish this, we propose Hadamard attention which introduces Hadamard product between queries and keys as the second-order term. In terms of formulation, Hadamard attention can be written as

$$\mathbf{h}_{j-i} = \text{Softmax}_j((\mathbf{q}_i \odot \mathbf{k}_i) \mathbf{r}_{j-i}^k + \mathbf{r}_{j-i}^q(\mathbf{q}_j \odot \mathbf{k}_j) + \mathbf{r}_{j-i}^b)) \quad (7)$$

where \odot means Hadamard product. With this simple yet effective modification, Equation 7 becomes a third-order representation of the input \mathbf{x} .

Besides being a third-order representation, Hadamard attention is easy to implement. Unlike Net7-N that needs memory- / time-consuming operations to calculate $\mathbf{q}_i \mathbf{k}_j$, $\mathbf{q}_i \odot \mathbf{k}_i$ and $\mathbf{q}_j \odot \mathbf{k}_j$ can be implemented via simple element-wise multiplication of the query and key feature maps. Also, \mathbf{r}_{j-i}^k and \mathbf{r}_{j-i}^q are equivalent to 1×1 convolutional filters. Thus, Equation 7 can be implemented by feature multiplication followed by convolution layers. For more implementation details please refer to the appendix and our code.

Ghost Head. In Section 3, we reveal the relation between the number of heads and model performance. Although channel setting is not the most critical factor that affects performance, more heads with better channel capacity still slightly boost the performance of DwConv-Swin-T in Figure 2. As directly enlarging the head number of LSA brings in no gain, we provide another cheap yet effective way to enrich channel capacity. Motivated by GhostNet [30], we propose the ghost head module which expands the original heads by combining them with two static matrices, represented as

$$\hat{h}_{j-i}^c = \text{Pow}(o_{j-i}^c, \lambda) h_{j-i}^{c'} + \gamma s_{j-i}^c \quad (8)$$

where $h_{j-i}^c / \hat{h}_{j-i}^c \in \mathbb{R}$ are the Hadamard attention values before / after the ghost head module, which are at c' / c -th channel and corresponds to the relative offset $j - i$; the relation between c and c' is $c' = c \% n_h$, where n_h is the number of heads and $\%$ is the mod operation; $O, S \in \mathbb{R}^{n_c \times ks \times ks}$ are two static matrices; o_{j-i}^c, s_{j-i}^c are the values in O, S at the c -th channel corresponding to the relative offset $j - i$; $\text{Pow}(\cdot)$ is the power operation; λ, γ are two hyper-parameters. The demo code of the ghost head is summarized in Algorithm 1. In the real implementation, we write CUDA operations to avoid large GPU memory consumption.

The ghost head is a cheap module. Its overhead is only $O(n_c \times ks \times ks \times n_p)$, where n_c is the number of channels, ks is the filter size (*i.e.* the size of neighboring areas), and n_p is the number of pixels. Recently, Refiner [91] is also proposed to adjust attention after softmax. Unlike them, the ghost head does not leverage heavy convolutions and linear

Algorithm 1 Demo code of ghost head (PyTorch-like)

```
# B: batch size, C: channel size
# N: the number of pixels
# H: the number of heads, K: kernel size
# h_attn: Hadamard attention with size (B, H, N, K*K)
# lambda, gamma: hyperparameters

def init():
    mul_matrix = nn.Parameters(torch.randn(C, K, K))
    add_matrix = nn.Parameters(torch.zeros(C, K, K))
    trunc_normal_(add_matrix, std=0.02)

def ghost_head(h_attn):
    # change the size of h_attn to (B, 1, H, N, K*K)
    h_attn = h_attn.unsqueeze(1)

    # reshape the size of matrices
    mul_matrix = mul_matrix.reshape(1, C//H, H, 1, K*K)
    add_matrix = add_matrix.reshape(1, C//H, H, 1, K*K)

    # Equation 8
    h_attn = (mul_matrix ** lambda) * h_attn + gamma *
        add_matrix
    return h_attn.reshape(B, C, N, K*K)
```

mappings, but only uses two simple static matrices. Also, the main purpose of the ghost head is not to refine attention, but to enrich channel capacity.

6. Experiments

We evaluate our ELSA in the Swin Transformer and VOLO architectures on ImageNet-1K image classification [19], COCO object detection [47], and ADE20K semantic segmentation [90]. For Swin Transformer [48], we drop-in replace LSA with ELSA without changing any architecture / hyperparameters. For VOLO [83], we adjust several hyperparameters of ELSA Blocks, while maintaining all other parts unchanged.

6.1. Image Classification on ImageNet-1K

Settings. Our major evaluations are conducted on the ImageNet-1K [19] dataset. During training, no extra training images are used. Our code is based on Pytorch [53], timm [72], DDF [92], Swin Transformer [48], and VOLO [83]. The detailed setups are as follows.

Swin Transformer. We replace LSA blocks of the first three stages with our ELSA blocks, and train ELSA-Swin following [48]. In particular, AdamW [49] is selected as the optimizer. The base learning rate is set to $1e-3$, which is scaled following the linear strategy, *i.e.* $lr = lr_{base} \times \frac{batch_size}{1024}$, and decays following the cosine strategy. We train models for 310 epochs, where the first 20 epochs are used for warm-up, and the last 10 epochs are used for cool-down. The weight decay of $5e-2$ is used. We also leverage the same augmentation and regularization strategies as [48]. Exponential moving average (EMA) [54, 61] is used in ELSA-Swin-B training. All models are trained / evaluated on 224×224 resolution unless otherwise specified.

VOLO. We replace all Outlooker modules in VOLO with

HA	GH	Params	FLOPs	Acc@1
<i>Base Model</i>		28.3M	4.5G	81.3
✓		29.0M	4.7G	82.4 ^{+1.1}
✓	✓	29.1M	4.8G	82.7^{+1.4}

Table 4. **Evaluate different components of ELSA.** The Swin-T is chosen as the baseline. HA means Hadamard attention, and GH denotes the ghost head module.

Architecture	Layer type	Params	FLOPs	Acc@1
Swin-T [48]	LSA [48]	28M	4.5G	81.3
	DwConv [32]	24M	3.7G	81.6
	D-DwConv [32]	51M	3.8G	81.9
	DDF [92]	46M	3.9G	82.0
	ELSA	29M	4.8G	82.7
VOLO-D1 [83]	LSA [83]	27M	–	83.8
	DwConv [83]	27M	–	83.8
	Outlooker [83]	27M	7.1G	84.2
	ELSA	27M	8.0G	84.7

Table 5. **Comparison of different layers.** Consistently boosts baselines and surpasses compared counterparts, while using over-head similar to LSA.

ELSA. We train ELSA-VOLOs following the training protocol of VOLO. Most settings of VOLO are similar to those of Swin Transformer, except for the following differences. The base learning rate is set to 1.6e-3 for VOLO-D1 and 8e-4 for VOLO-D5. The Token Labeling [40] is used during training, thus, MixUp [86] and CutMix [85] are replaced by MixToken [40]. EMA is used in ELSA-VOLO-D5 training. Please refer to [83] for more details.

Ablation study. We respectively analyze the effect of Hadamard attention and the ghost head module in ELSA. We choose Swin-T [48] as our base architecture and experiment with different modifications to ELSA. Table 4 shows the results of ablation experiments. The performance of Swin-T is improved by 1.1%, with only Hadamard attention. We observed another 0.3% improvements by plugging-in the ghost head.

Table 5 further compares the performance between different types of layers on both Swin Transformer and VOLO. As can be seen, Swin-T and VOLO-D1 with ELSA respectively achieve 82.7% and 84.7% top-1 accuracy, which surpass other compared counterparts.

Compare with the state-of-the-art models. We compare ELSA-Swin and ELSA-VOLO with other state-of-the-art models in Table 6. #Res represents resolutions used in validating / finetuning. For fair comparisons, results are split into groups according to the number of parameters.

As can be seen, for different model sizes, our proposed ELSA consistently boosts Swin Transformer and VOLO,

Model	Params	FLOPs	#Res	Acc@1
T2T-ViT-14 [82]	22M	5.2G	224 ²	81.5
CoAtNet-0 [17]	25M	4.2G	224 ²	81.6
Twins-SVT-S [14]	24M	2.9G	224 ²	81.7
Swin-T [48]	28M	4.5G	224 ²	81.3
VOLO-D1 [83]	27M	7.1G	224 ²	84.2
VOLO-D1 _{↑384} [83]	27M	20.8G	384 ²	85.2
ELSA-Swin-T	28M	4.8G	224²	82.7
ELSA-VOLO-D1	27M	8.0G	224²	84.7
ELSA-VOLO-D1_{↑384}	27M	23.3G	384²	85.7
T2T-ViT-24 [82]	64M	15.0G	224 ²	82.6
CoAtNet-1 [17]	42M	8.4G	224 ²	83.3
Twins-SVT-B [14]	56M	8.6G	224 ²	83.2
Swin-S [48]	50M	8.7G	224 ²	83.0
ELSA-Swin-S	53M	9.6G	224²	83.5
CoAtNet-2 [17]	75M	15.7G	224 ²	84.1
Twins-SVT-L [14]	99M	15.1G	224 ²	83.7
Swin-B [48]	88M	15.4G	224 ²	83.5
VOLO-D3 [83]	86M	20.9G	224 ²	85.4
VOLO-D3 _{↑448} [83]	86M	92.9G	448 ²	86.3
ELSA-Swin-B	93M	16.7G	224²	84.0
ELSA-VOLO-D3	87M	22.3G	224²	85.7
ELSA-VOLO-D3_{↑448}	87M	98.6G	448²	86.6
CoAtNet-3 [17]	168M	34.7G	224 ²	84.5
VOLO-D4 [83]	193M	44.6G	224 ²	85.7
VOLO-D4 _{↑448} [83]	193M	194G	448 ²	86.8
CaiT-M36 _{↑448} [63]	271M	248G	448 ²	86.3
CaiT-M48 _{↑448} [63]	356M	330G	448 ²	86.5
VOLO-D5 [83]	295M	72.7G	224 ²	86.1
VOLO-D5 _{↑512} [83]	295M	407G	512 ²	87.1
ELSA-VOLO-D5	298M	78.5G	224²	86.3
ELSA-VOLO-D5_{↑512}	298M	437G	512²	87.2

Table 6. **Comparison of different backbones on the ImageNet-1K.** Simply plugging-in ELSA achieves state-of-the-art performance.

while introducing little overhead. In particular, ELSA improves Swin-T, Swin-S, and Swin-B by 1.4%, 0.5%, and 0.5%, where ELSA-Swin-S is comparable to the original Swin-B with two-thirds of parameters and FLOPs. Testing on the resolution of 224, ELSA-VOLO-D1 and ELSA-VOLO-D3 yield 84.7% and 85.7% top-1 accuracy, respectively. The performance of ELSA-VOLO-D3 matches the performance of the original VOLO-D4. However, VOLO-D4 costs more than 2× parameters against ELSA-VOLO-D3. Without additional images, it is very difficult to improve the accuracy of large models under supervised training due to over-fitting. Even so, ELSA still slightly improves VOLO-D5.

It is worth noting that all these records of ELSA-Swin are obtained without modifying any hyperparameters in Swin Transformers, which may not be the optimal setting for our ELSA. For VOLO, we also keep the structure and hyper-

Mask RCNN with $1\times$ schedule								
Backbone	AP ^b	AP ₅₀ ^b	AP ₇₅ ^b	AP ^m	AP ₅₀ ^m	AP ₇₅ ^m	Params	FLOPs
PVT-M [70]	42.0	64.4	45.6	39.0	61.6	42.1	64M	–
Region-S [6]	44.2	67.3	48.2	40.8	64.1	44.0	51M	183G
Focal-T [79]	44.8	–	–	41.0	–	–	49M	291G
ViL-S [87]	44.9	67.1	49.3	41.0	64.2	44.1	45M	218G
Swin-T [48]	43.7	66.6	47.7	39.8	63.3	42.7	48M	267G
ELSA-T	45.6	67.9	50.3	41.1	64.8	44.0	49M	269G
PVT-L [70]	42.9	65.0	46.6	39.5	61.9	42.5	81M	–
Region-B [6]	46.3	69.1	51.2	42.4	66.2	45.6	93M	464G
Focal-S [79]	47.4	–	–	42.8	–	–	71M	401G
ViL-M [87]	47.6	69.8	52.1	43.0	66.9	46.6	60M	294G
Swin-S [†] [48]	46.5	–	–	42.1	–	–	69M	354G
ELSA-S	48.3	70.4	52.9	43.0	67.3	46.4	72M	367G

Cascade Mask RCNN with $1\times$ schedule								
Backbone	AP ^b	AP ₅₀ ^b	AP ₇₅ ^b	AP ^m	AP ₅₀ ^m	AP ₇₅ ^m	Params	FLOPs
Swin-T [48]	48.1	67.1	52.2	41.7	64.4	45.0	86M	745G
ELSA-T	49.8	68.9	54.2	43.0	66.1	46.3	86M	748G
Swin-S [48]	50.3	69.7	54.4	43.4	67.0	46.7	107M	838G
ELSA-S	51.6	70.5	56.0	44.4	67.8	47.8	110M	846G

Table 7. **Comparison of different backbones on the COCO validation set.** AP^b / AP^m denote the mean average precision of detection / segmentation. [†] denotes the results are reported in [79].

parameters of other parts unchanged. Under such control variable principle, the use of ELSA blocks still achieves state-of-the-arts. Redesigning the macro architecture / hyperparameters manually or by NAS may yield better Pareto performance.

6.2. Object Detection on COCO

Settings. Object detection and instance segmentation experiments are conducted on the COCO dataset. We report the performance on the validation subset, and use the mean average precision (AP) as the metric. We evaluate ELSA-Swin in Mask RCNN / Cascade Mask RCNN [2, 33], which is a common practice in [6, 70, 71, 79, 87]. Following the common training protocol, we apply multi-scale training, scaling the shorter side of the input from 480 to 800 while keeping the longer side no more than 1333. AdamW [49] is adopted as the optimizer with an initial learning rate of $1e-4$, weight decay of $5e-2$, and batch size of 16. For fair comparisons, all backbones are pretrained using the ImageNet-1K only, and finetuned on the COCO with $1\times$ schedule (12 epochs). Our implementation is based on Swin Transformer [48] and mmdetection [8].

Results. Table 7 lists experimental results. As can be seen, ELSA-Swin-T and ELSA-Swin-S (noted as ELSA-T / ELSA-S) respectively improve the corresponding baseline by 1.9 AP and 1.8 AP in detection, both outperforming other methods within their group. Note that, unlike ViL [87] and RegionViT [6], ELSA-Swin does not modify the macro

Backbone	Params	FLOPs	MS mIoU
Swin-T [48]	60M	945G	45.8
Focal-T [79]	62M	998G	47.0
Twins-SVT-S [14]	54M	–	47.1
ELSA-Swin-T	61M	946G	47.7_{+1.9}
Swin-S [62]	81M	1038G	49.5
Focal-S [79]	85M	1130G	50.0
Twins-SVT-B [14]	89M	–	48.9
ELSA-Swin-S	85M	1046G	50.3_{+0.8}
Swin-B [48]	121M	1188G	49.7
Focal-B [79]	126M	1354G	50.5
Twins-SVT-L [14]	133M	–	50.2

Table 8. **Comparison of different backbones on the ADE20K validation set.** UperNet [74] is adopted as the framework. All compared backbones are pretrained with the ImageNet-1K only.

architecture / hyperparameters of Swin Transformer. Cascade Mask RCNNs with ELSA-Swin-T and ELSA-Swin-S achieve 49.8 AP and 51.6 AP in detection, which are 1.7 AP and 1.3 AP higher than their baselines. The appendix shows more comparisons with $3\times$ schedule on the COCO dataset.

6.3. Semantic Segmentation on ADE20K

Settings. We evaluate the semantic segmentation performance of ELSA-Swin on the ADE20K [90], which contains 20K training, 2K validation, and 3K testing images, covering 150 semantic categories. Following [32, 48, 83], UperNet [74] is selected as the baseline framework. During training, AdamW is adopted as the optimizer. The initial learning rate is set to $6e-5$, weight decay is set to $1e-2$. All models are trained for 160K iterations with linear learning rate decay, and a linear warmup of 1500 iterations. We use default augmentation settings in mmsegmentation [16] where the resolution of the input is set to 512×512 . During inference, we perform multi-scale test with interpolation rates of [0.75, 1.0, 1.25, 1.5, 1.75]. For more details, please refer to [16, 48] and our code.

Results. Table 8 shows the mean IoU with multi-scale testing (MS mIoU), model size (Param), and FLOPs of different methods. Results are split into three groups based on the number of model parameters. For fair comparisons, all compared backbones are pretrained using the ImageNet-1K only. UperNet with ELSA-Swin-T is 1.9 higher on MS mIoU than the Swin-T version. Adopting ELSA-Swin-S as the backbone achieves 50.3 MS mIoU, which is 0.8 higher on MS mIoU than Swin-S, and is even better than Swin-B and Twins-SVT-L in the third group.

7. Conclusions

In this work, we investigate LSA and its counterparts in detail from channel settings and spatial processing to em-

pirically understand why LSA performs mediocre. It is revealed that the relative position embedding and the neighboring filter application are critical reasons why DwConv and dynamic filters perform similar or better than LSA. Based on these observations, we further propose the enhanced local self-attention (ELSA) with Hadamard Attention and the ghost head, which can seamlessly replace LSA and its counterparts in various networks. Experiments show that, without other architecture / hyperparameter modifications, ELSA can consistently improve the baseline, regardless of the model size and tasks, with little overhead being introduced.

Acknowledgements

This work was supported by Alibaba Group through Alibaba Research Intern Program and the National Natural Science Foundation of China (No.61976094).

References

- [1] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, 2020. 2
- [2] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *CVPR*, 2018. 8
- [3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 1
- [4] Boyu Chen, Peixia Li, Baopu Li, Chuming Li, Lei Bai, Chen Lin, Ming Sun, Junjie Yan, and Wanli Ouyang. Psvit: Better vision transformer via token pooling and attention sharing. *arXiv preprint arXiv:2108.03428*, 2021. 2
- [5] Chun-Fu Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. *arXiv preprint arXiv:2103.14899*, 2021. 2
- [6] Chun-Fu Chen, Rameswar Panda, and Quanfu Fan. Regionvit: Regional-to-local attention for vision transformers. *arXiv preprint arXiv:2106.02689*, 2021. 8
- [7] Jin Chen, Xijun Wang, Zichao Guo, Xiangyu Zhang, and Jian Sun. Dynamic region-aware convolution. In *CVPR*, 2021. 2
- [8] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Openmmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 8
- [9] Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Xiaoyi Dong, Lu Yuan, and Zicheng Liu. Mobileformer: Bridging mobilenet and transformer. *arXiv preprint arXiv:2108.05895*, 2021. 2
- [10] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *CVPR*, 2020. 2
- [11] Zhengsu Chen, Lingxi Xie, Jianwei Niu, Xuefeng Liu, Longhui Wei, and Qi Tian. Visformer: The vision-friendly transformer. In *ICCV*, 2021. 1, 2
- [12] Zhiyang Chen, Yousong Zhu, Chaoyang Zhao, Guosheng Hu, Wei Zeng, Jinqiao Wang, and Ming Tang. Dpt: Deformable patch-based transformer for visual recognition. In *ACM MM*, pages 2899–2907, 2021. 2
- [13] François Chollet. Xception: Deep learning with depthwise separable convolutions. *PAMI*, 2017. 1, 2
- [14] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. In *NeurIPS*, 2021. 7, 8
- [15] Xiangxiang Chu, Bo Zhang, Zhi Tian, Xiaolin Wei, and Huaxia Xia. Do we really need explicit position encodings for vision transformers? *arXiv preprint arXiv:2102.10882*, 2021. 2
- [16] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. <https://github.com/open-mmlab/mmdetection>, 2020. 8
- [17] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *arXiv preprint arXiv:2106.04803*, 2021. 1, 2, 7
- [18] 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. 2
- [19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 6
- [20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019. 2
- [21] 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. 2
- [22] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2020. 1, 2
- [23] Alaaeldin El-Nouby, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, et al. Xcit: Cross-covariance image transformers. *arXiv preprint arXiv:2106.09681*, 2021. 2
- [24] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. *arXiv preprint arXiv:2104.11227*, 2021. 2

- [25] Yuxin Fang, Xinggang Wang, Rui Wu, and Wenyu Liu. What makes for hierarchical vision transformer? *arXiv preprint arXiv:2107.02174*, 2021. 1, 2
- [26] Chengyue Gong, Dilin Wang, Meng Li, Vikas Chandra, and Qiang Liu. Improve vision transformers training by suppressing over-smoothing. *arXiv preprint arXiv:2104.12753*, 2021. 2
- [27] Ben Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: A vision transformer in convnet’s clothing for faster inference. *arXiv preprint arXiv:2104.01136*, 2021. 2
- [28] Jiaqi Gu, Hyoukjun Kwon, Dilin Wang, Wei Ye, Meng Li, Yu-Hsin Chen, Liangzhen Lai, Vikas Chandra, and David Z Pan. Hrvit: Multi-scale high-resolution vision transformer. *arXiv preprint arXiv:2111.01236*, 2021. 2
- [29] Jianyuan Guo, Kai Han, Han Wu, Chang Xu, Yehui Tang, Chunjing Xu, and Yunhe Wang. Cmt: Convolutional neural networks meet vision transformers. *arXiv preprint arXiv:2107.06263*, 2021. 2
- [30] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *CVPR*, 2020. 6
- [31] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *arXiv preprint arXiv:2103.00112*, 2021. 2
- [32] Qi Han, Zejia Fan, Qi Dai, Lei Sun, Ming-Ming Cheng, Jiaying Liu, and Jingdong Wang. Demystifying local vision transformer: Sparse connectivity, weight sharing, and dynamic weight. *arXiv preprint arXiv:2106.04263*, 2021. 1, 2, 7, 8
- [33] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *CVPR*, 2017. 8
- [34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2
- [35] Shuting He, Hao Luo, Pichao Wang, Fan Wang, Hao Li, and Wei Jiang. Transreid: Transformer-based object re-identification. In *ICCV*, 2021. 1
- [36] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, 2019. 2
- [37] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1, 2
- [38] Zilong Huang, Youcheng Ben, Guozhong Luo, Pei Cheng, Gang Yu, and Bin Fu. Shuffle transformer: Rethinking spatial shuffle for vision transformer. *arXiv preprint arXiv:2106.03650*, 2021. 2
- [39] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *NeurIPS*, 2016. 2
- [40] Zihang Jiang, Qibin Hou, Li Yuan, Daquan Zhou, Xiaojie Jin, Anran Wang, and Jiashi Feng. Token labeling: Training a 85.5% top-1 accuracy vision transformer with 56m parameters on imagenet. *arXiv preprint arXiv:2104.10858*, 2021. 7
- [41] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020. 2
- [42] Duo Li, Jie Hu, Changhu Wang, Xiangtai Li, Qi She, Lei Zhu, Tong Zhang, and Qifeng Chen. Involution: Inverting the inheritance of convolution for visual recognition. In *CVPR*, 2021. 2, 3, 4
- [43] Jinpeng Li, Yichao Yan, Shengcai Liao, Xiaokang Yang, and Ling Shao. Local-to-global self-attention in vision transformers. *arXiv preprint arXiv:2107.04735*, 2021. 2
- [44] Shanda Li, Xiangning Chen, Di He, and Cho-Jui Hsieh. Can vision transformers perform convolution? *arXiv preprint arXiv:2111.01353*, 2021. 2
- [45] Yawei Li, Kai Zhang, Jie Zhang Cao, Radu Timofte, and Luc Van Gool. Localvit: Bringing locality to vision transformers. *arXiv preprint arXiv:2104.05707*, 2021. 2
- [46] Ming Lin and Jieping Ye. A non-convex one-pass framework for generalized factorization machine and rank-one matrix sensing. In *NeurIPS*, 2016. 6
- [47] 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. 6
- [48] 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. In *ICCV*, 2021. 1, 2, 3, 4, 6, 7, 8, 12
- [49] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 6, 8
- [50] Jiachen Lu, Jinghan Yao, Junge Zhang, Xiatian Zhu, Hang Xu, Weiguo Gao, Chunjing Xu, Tao Xiang, and Li Zhang. Soft: Softmax-free transformer with linear complexity. *arXiv preprint arXiv:2110.11945*, 2021. 2
- [51] Ningning Ma, Xiangyu Zhang, Jiawei Huang, and Jian Sun. Weightnet: Revisiting the design space of weight networks. In *ECCV*, 2020. 2
- [52] Sachin Mehta and Mohammad Rastegari. Mobilevit: Lightweight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021. 2
- [53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 6
- [54] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 1992. 6
- [55] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? In *NeurIPS*, 2021. 1, 2
- [56] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *arXiv preprint arXiv:2106.02034*, 2021. 2
- [57] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 2

- [58] Hang Su, Varun Jampani, Deqing Sun, Orazio Gallo, Erik Learned-Miller, and Jan Kautz. Pixel-adaptive convolutional neural networks. In *CVPR*, 2019. 2
- [59] Peize Sun, Yi Jiang, Rufeng Zhang, Enze Xie, Jinkun Cao, Xinting Hu, Tao Kong, Zehuan Yuan, Changhu Wang, and Ping Luo. Transtrack: Multiple-object tracking with transformer. *arXiv preprint arXiv:2012.15460*, 2020. 1
- [60] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 2
- [61] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *arXiv preprint arXiv:1703.01780*, 2017. 6
- [62] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021. 1, 8, 12
- [63] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *arXiv preprint arXiv:2103.17239*, 2021. 7
- [64] Ashish Vaswani, Prajit Ramachandran, Aravind Srinivas, Niki Parmar, Blake Hechtman, and Jonathon Shlens. Scaling local self-attention for parameter efficient visual backbones. In *CVPR*, 2021. 2
- [65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 2
- [66] Jiaqi Wang, Kai Chen, Rui Xu, Ziwei Liu, Chen Change Loy, and Dahua Lin. Carafe: Content-aware reassembly of features. In *ICCV*, 2019. 2, 3
- [67] Jiaqi Wang, Kai Chen, Rui Xu, Ziwei Liu, Chen Change Loy, and Dahua Lin. Carafe++: Unified content-aware reassembly of features. *PAMI*, 2021. 2, 3
- [68] Pichao Wang, Xue Wang, Hao Luo, Jinglai Zhou, Zhipeng Zhou, Fan Wang, Hao Li, and Rong Jin. Scaled relu matters for training vision transformers. *arXiv preprint arXiv:2109.03810*, 2021. 2
- [69] Pichao Wang, Xue Wang, Fan Wang, Ming Lin, Shuning Chang, Wen Xie, Hao Li, and Rong Jin. Kvt: K-nn attention for boosting vision transformers. *arXiv preprint arXiv:2106.00515*, 2021. 2
- [70] 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. In *ICCV*, 2021. 8, 12
- [71] Wenxiao Wang, Lu Yao, Long Chen, Deng Cai, Xiaofei He, and Wei Liu. Crossformer: A versatile vision transformer based on cross-scale attention. *arXiv preprint arXiv:2108.00154*, 2021. 8
- [72] Ross Wightman. Pytorch image models. <https://github.com/rwightman/pytorch-image-models>, 2019. 6
- [73] Kan Wu, Houwen Peng, Minghao Chen, Jianlong Fu, and Hongyang Chao. Rethinking and improving relative position encoding for vision transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10033–10041, 2021. 2
- [74] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *ECCV*, 2018. 8
- [75] 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. 2
- [76] Weijian Xu, Yifan Xu, Tyler Chang, and Zhuowen Tu. Co-scale conv-attentional image transformers. *arXiv preprint arXiv:2104.06399*, 2021. 2
- [77] Yifan Xu, Zhijie Zhang, Mengdan Zhang, Kekai Sheng, Ke Li, Weiming Dong, Liqing Zhang, Changsheng Xu, and Xing Sun. Evo-vit: Slow-fast token evolution for dynamic vision transformer. *arXiv preprint arXiv:2108.01390*, 2021. 2
- [78] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. In *NeurIPS*, 2019. 2
- [79] Jianwei Yang, Chunyuan Li, Pengchuan Zhang, Xiyang Dai, Bin Xiao, Lu Yuan, and Jianfeng Gao. Focal self-attention for local-global interactions in vision transformers. *arXiv preprint arXiv:2107.00641*, 2021. 2, 8, 12
- [80] Qihang Yu, Yingda Xia, Yutong Bai, Yongyi Lu, Alan Yuille, and Wei Shen. Glance-and-gaze vision transformer. *arXiv preprint arXiv:2106.02277*, 2021. 2
- [81] Kun Yuan, Shaopeng Guo, Ziwei Liu, Aojun Zhou, Fengwei Yu, and Wei Wu. Incorporating convolution designs into visual transformers. *arXiv preprint arXiv:2103.11816*, 2021. 2
- [82] 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. In *ICCV*, 2021. 1, 2, 7
- [83] Li Yuan, Qibin Hou, Zihang Jiang, Jiashi Feng, and Shuicheng Yan. Volo: Vision outlooker for visual recognition. *arXiv preprint arXiv:2106.13112*, 2021. 1, 2, 3, 4, 6, 7, 8
- [84] Yuhui Yuan, Rao Fu, Lang Huang, Weihong Lin, Chao Zhang, Xilin Chen, and Jingdong Wang. Hrformer: High-resolution transformer for dense prediction. *NeurIPS*, 2021. 2
- [85] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *CVPR*, 2019. 7
- [86] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. Mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 7
- [87] Pengchuan Zhang, Xiyang Dai, Jianwei Yang, Bin Xiao, Lu Yuan, Lei Zhang, and Jianfeng Gao. Multi-scale vision longformer: A new vision transformer for high-resolution image encoding. In *ICCV*, 2021. 1, 2, 4, 5, 8, 12
- [88] Yikang Zhang, Jian Zhang, Qiang Wang, and Zhao Zhong. Dynet: Dynamic convolution for accelerating convolutional neural networks. *arXiv preprint arXiv:2004.10694*, 2020. 2
- [89] 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 *CVPR*, 2021. 1

- [90] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017. 6, 8
- [91] Daquan Zhou, Yujun Shi, Bingyi Kang, Weihao Yu, Zihang Jiang, Yuan Li, Xiaojie Jin, Qibin Hou, and Jiashi Feng. Refiner: Refining self-attention for vision transformers. *arXiv preprint arXiv:2106.03714*, 2021. 6
- [92] Jingkai Zhou, Varun Jampani, Zhixiong Pi, Qiong Liu, and Ming-Hsuan Yang. Decoupled dynamic filter networks. In *CVPR*, 2021. 1, 2, 3, 4, 6, 7, 12

A. Additional Investigation

In addition to peak performance, we also compared the convergence speed between Filter Normalization [92] and softmax. We select Net7 as our baseline and show the accuracy at different epochs in Table 9. As can be seen, the Filter Normalization can speed up model convergence at early epochs. However, due to the constraint of 0 mean and 1 variance in Filter Normalization, the flexibility of filters are limited. Therefore, in the last 100 epochs, Net7 with Filter Normalization lags behind the softmax version.

B. COCO Detection with $3\times$ Schedule

Table 10 lists the COCO detection results with $3\times$ schedule (36 epochs). As can be seen, ELSA-Swin-T (noted as ELSA-T) improves the baseline by 1.5 box AP / 1.1 mask AP; ELSA-Swin-S (noted as ELSA-S) boosts the baseline by 0.7 box AP / 0.3 mask AP. Both of their box AP surpass other counterparts within their group. Cascade Mask RCNNs with ELSA-Swin-T and ELSA-Swin-S achieve 51.1 AP and 52.3 AP in detection, which are 0.6 AP and 0.5 AP higher than their baselines. It is worth noting that the ELSA-Swin-S version even surpasses the the Swin-B one.

C. Ghost Head on Global Self-Attention

Beyond ELSA, we also evaluate the performance of the ghost head when applied to global self-attention. We chose DeiT-Ti [62] as the baseline. We first remove the class / distillation tokens in DeiT-Ti and use the global average pooling to generate the feature for classification. Then, we apply the ghost head module to global self-attention maps after softmax. Instead of expanding the number of heads to the number of channels, in the global self-attention case, the ghost head module only doubles the number of heads. As can be seen in Table 11, the ghost head module also considerably improves the the baseline by 3.7% on top-1 accuracy.

Epochs	50	100	150	200	250	300
Filter Norm [92]	71.5	74.9	77.2	79.4	80.8	81.4
Softmax	70.3	74.7	77.1	79.3	81.0	81.8

Table 9. **Top-1 Accuracy at different training epochs.** Filter normalization speeds up the model convergence at early epochs, but falls into the local minimum due to constraints.

Mask RCNN with $3\times$ schedule								
Backbone	AP ^b	AP ₅₀ ^b	AP ₇₅ ^b	AP ^m	AP ₅₀ ^m	AP ₇₅ ^m	Params	FLOPs
PVT-M [70]	44.2	66.0	48.2	40.5	63.1	43.5	64M	—
Focal-T [79]	47.2	69.4	51.9	42.7	66.5	45.9	49M	291G
ViL-S [87]	47.1	68.7	51.5	42.7	65.9	46.2	45M	218G
Swin-T [48]	46.0	68.1	50.3	41.6	65.1	44.9	48M	267G
ELSA-T	47.5	69.1	52.3	42.7	66.3	45.9	49M	269G
PVT-L [70]	44.5	66.0	48.3	40.7	63.4	43.7	81M	—
Focal-S [79]	48.8	70.5	53.6	43.8	67.7	47.2	71M	401G
ViL-M [87]	48.9	70.3	54.0	44.2	67.9	47.7	60M	294G
Swin-S [†] [48]	48.5	70.2	53.5	43.3	67.3	46.6	69M	354G
ELSA-S	49.2	70.3	54.3	43.6	67.4	46.8	72M	367G
Cascade Mask RCNN with $3\times$ schedule								
Backbone	AP ^b	AP ₅₀ ^b	AP ₇₅ ^b	AP ^m	AP ₅₀ ^m	AP ₇₅ ^m	Params	FLOPs
Swin-T [48]	50.5	69.3	54.9	43.7	66.6	47.1	86M	745G
ELSA-T	51.1	69.7	55.3	44.2	67.2	48.1	86M	748G
Swin-S [48]	51.8	70.4	56.3	44.7	67.9	48.5	107M	838G
ELSA-S	52.3	70.9	57.1	45.2	68.4	49.2	110M	846G
Swin-B [48]	51.9	70.9	56.5	45.0	68.4	48.7	145M	982G

Table 10. **Comparison of different backbones on the COCO validation set.** AP^b / AP^m denote the mean average precision of detection / segmentation. [†] denotes the results are reported in [79].

Models	Params	#Res	Acc@1	Acc@5
DeiT-Ti Original [62]	6M	224 ²	72.2	91.1
DeiT-Ti GAP	6M	224 ²	73.8	92.2
DeiT-Ti GAP w/ GH	6M	224 ²	77.5	93.7

Table 11. **Evaluation of applying ghost head to global self-attention.** GAP means global average pooling, GH denotes the ghost head module.

D. Implementation of Hadamard Attention

We do not implement Hadamard attention strictly according to Equation 7. Hadamard attention is implemented as a variant of Equation ?? which is faster and better. Here, we will show how to get our implementation step by step from Equation 7 in the form of pseudo code.

The strict implementation of Equation 7 is shown in Algorithm 2, where the unfold operation is used to calculate $\mathbf{q}_j \odot \mathbf{k}_j$. The unfold operation is memory-consuming, while the main function of it is to shift the feature map and align the pixel i and j , so that we can directly add $\mathbf{q}_i \odot \mathbf{k}_i$ with $\mathbf{q}_j \odot \mathbf{k}_j$. Instead of using unfold operation, we can

Algorithm 2 Strictly follow Equation 7 (PyTorch-like)

```
# B: batch size, C: channel size
# H, W: the height / width of feature map
# G: the number of heads, K: kernel size
# q, k: queries and keys in shape (B, C, H, W)

def init():
    rq = nn.Parameters(torch.randn(C, G, K*K))
    trunc_normal_(rq, std=0.02)
    rk = nn.Parameters(torch.zeros(C, G, K*K))
    trunc_normal_(rk, std=0.02)
    rb = nn.Parameters(G, K*K)
    trunc_normal_(rb, std=0.02)
    unfold_op = nn.Unfold(kernel_size=window_size,
        padding=window_size//2, stride=1)

def cal_hp_rk(hp):
    return torch.einsum('bchw,cgk->bgkhw', hp, rk)

def cal_rq_hp(hp):
    hp = unfold_op(hp).reshape(B, C, K*K, H, W)
    return torch.einsum('cgk,bckhw->bgkhw', rq, hp)

def cal_h_attn(q, k):
    # Hadamard product
    hp = q*k
    # (B, G, K*K, H, W)
    return (cal_hp_rk(hp) + cal_rq_hp(hp) + rb.reshape(
        1, G, K*K, 1, 1)).softmax(2)
```

use depth-wise convolution to shift feature maps, which is shown in Algorithm 3.

Algorithm 3 can be further simplified by merge two 1×1 convolution layers to one layer, as shown in Algorithm 4. Note that Algorithm 2, Algorithm 3, and Algorithm 4 are completely equivalent. One can get the same output by setting the same model parameters.

As can be seen in Algorithm 4, Equation 7 can be implemented by feature multiplication followed with a 1×1 convolution and a group convolution. For our final implementation (a variant of Equation 7), we slightly modify the sequence of these convolutional layers and add an activation function (GELU) between them, which is shown in the attached code.

Algorithm 3 Equivalent 1 of Equation 7 (PyTorch-like)

```
# B: batch size, C: channel size
# H, W: the height / width of feature map
# G: the number of heads, K: kernel size
# q, k: queries and keys in shape (B, C, H, W)

def init():
    # einsum is equivalent to 1x1 convolution
    # rb is equivalent to bias in convolution
    cal_hp_rk = nn.Conv2d(C, G*K*K, 1, bias=True)
    cal_rq_hp = nn.Conv2d(C, G*K*K, 1, bias=True)

def cal_h_attn(q, k):
    # Hadamard product
    hp = q*k

    hp_rk = cal_hp_rk(hp)
    rq_hp = cal_rq_hp(hp)

    kernel = torch.zeros(G*K*K, 1, K, K)
    for i in range(G*K*K):
        _id = i % K*K
        _x = _id % K
        _y = _id // K
        kernel[i, 0, _y, _x] = 1
    rq_hp = F.conv2d(rq_hp, kernel, padding=K//2,
        groups=G*K*K)
    rq_hp = rq_hp

    h_attn = (hp_rk + rq_hp).reshape(B, G, K*K, H, W)
    return h_attn.softmax(2)
```

Algorithm 4 Equivalent 2 of Equation 7 (PyTorch-like)

```
# B: batch size, C: channel size
# H, W: the height / width of feature map
# G: the number of heads, K: kernel size
# q, k: queries and keys in shape (B, C, H, W)

def init():
    # merge layers
    conv_1x1 = nn.Conv2d(C, 2*G*K*K, 1, bias=True)

def cal_h_attn(q, k):
    # Hadamard product
    hp = q*k

    hp_attn = conv_1x1(hp)

    kernel = torch.zeros(G*K*K, 2, K, K)
    for i in range(G*K*K):
        _id = i % K*K
        _x = _id % K
        _y = _id // K
        kernel[i, 0, K//2, K//2] = 1
        kernel[i, 1, _y, _x] = 1
    hp_attn = F.conv2d(hp_attn, kernel, padding=K//2,
        groups=G*K*K)
    return hp_attn.reshape(B, G, K*K, H, W).softmax(2)
```
