Pruning Self-attentions into Convolutional Layers in Single Path

Haoyu He[1] Jing Liu[1] Zizheng Pan[1] Jianfei Cai[1]
Jing Zhang[2] Dacheng Tao[3] Bohan Zhuang[1†]

[1]Monash University [2]The University of Sydney [3]JD Explore Academy

Abstract

*Vision Transformers (ViTs) have achieved impressive performance over various computer vision tasks. However, modeling global correlations with multi-head self-attention (MSA) layers leads to two widely recognized issues: the massive computational resource consumption and the lack of intrinsic inductive bias for modeling local visual patterns. One unified solution is to search whether to replace some MSA layers with convolution-like inductive biases that are computationally efficient via neural architecture search (NAS) based pruning methods. However, maintaining MSA and different candidate convolutional operations as separate trainable paths gives rise to expensive search cost and challenging optimization. Instead, we propose a novel weight-sharing scheme between MSA and convolutional operations and cast the search problem as finding which subset of parameters to use in each MSA layer. The weight-sharing scheme further allows us to devise an automatic **S**ingle-**P**ath **Vi**sion **T**ransformer pruning method (SPViT) to quickly prune the pre-trained ViTs into accurate and compact hybrid models with significantly reduced search cost, given target efficiency constraints. We conduct extensive experiments on two representative ViT models showing our method achieves a favorable accuracy-efficiency trade-off. Code is available at* https://github.com/zhuang-group/SPViT.

1. Introduction

Vision Transformers [18, 41, 48] have attracted substantial research interests and become one of the dominant backbones in various image recognition tasks, such as classification [54], segmentation [65] and detection [7].

Despite the huge excitement generated by the recent development, two limitations of ViTs introduced by multi-head self-attention layers [56] have been recognized. Firstly, a well-known concern with MSA layers is the quadratic time and memory complexity, hindering the development and deployment of ViTs at scale, especially for modeling long sequences. To this end, many efforts have been made to develop efficient Transformers, *e.g.*, pruning methods [11, 34, 67] are employed to prune less important ViT components. Another fundamental problem is that MSA layers lack a locality modeling mechanism for encoding local information [37], which is essential for handling low-level image patterns, *e.g.*, edges and shapes. To address the issue, prior arts propose to introduce inductive bias by inserting convolutional layers in ViTs with various heuristics, *e.g.*, inside feed-forward network layers (FFNs) [23, 37], before the ViT encoder [59] or before the MSA layers [61].

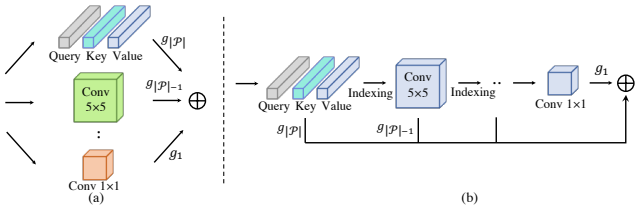

Figure 1. Multi-path vs. single-path search space. (a) Multi-path search space. MSA and different types of convolutional operations are added as separate trainable paths, leading to an expensive search cost. $\mathcal{P}$ and g_p denote the ordered operation set and the learnable gate for the p-th operation, respectively. (b) Single-path search space. We propose to formulate searching as a subset selection problem where the outputs of the convolutional operations are directly indexed from the MSA intermediate results, thereby easing the optimization difficulty and reducing the search cost together with the number of trainable parameters.

One unified solution to solve both issues is to automatically search whether to replace some MSA layers modeling the global correlations with local convolutional inductive bias, following NAS-based pruning methods [26, 63]. As a result, we can eliminate some costly MSA layers to significantly improve the efficiency for ViT models while adding proper locality without heuristics. Recently, several one-shot NAS methods propose to include candidate MSA and convolutional operations separately to the search space [9, 33, 60], where each operation is formulated as a separate trainable path (see Figure 1 (a)). However, these *multi-path* methods can give rise to the expensive search cost and challenging optimization since all the candidate operations for each layer need to be maintained and updated

†Corresponding author. Email: bohan.zhuang@monash.edu

independently during the search.

To tackle this limitation, we propose a novel **S**ingle-**P**ath **Vi**sion **T**ransformer pruning approach, called SPViT, to efficiently prune the pre-trained ViTs into compact models with low search cost and high accuracy. Specifically, inspired by the observation that MSA layers have the capacity for modeling local regions and MSA heads sometimes attend to local regions [14, 50], we develop a weight-sharing scheme between MSA and convolutional operations. Consequently, we can obtain good initialization of the convolution kernel weights using pre-trained MSA parameters and derive the outputs for the convolutional operations by indexing MSA intermediate results easily during the search. By further encapsulating all the candidate operations into a single MSA per layer, we form a *single-path* search scheme as shown in Figure 1 (b). Without having to choose among different operations as in *multi-path* methods, we instead formulate the search problem as finding which subset of weights to use in each MSA layer thereby pruning the MSA into convolutional layers when convolutional layers are selected. Our search scheme can therefore deliver a reduced number of parameters, computational cost, and optimization difficulty.

Additionally, considering FFN layers consume a significant proportion of computations, *e.g.*, 11.1G out of 17.5G Mult-Adds when processing 224×224 images with DeiT base model [54], we propose a simple but effective way to search for fine-grained MLP expansion ratios customizing the number of hidden dimensions for each FFN layer. Specifically, our SPViT employs learnable gates to determine the importance of each FFN hidden dimension, and then prunes the less-important dimensions.

Our main contributions can be summarized as follows:

- We propose a novel weight-sharing scheme between the MSA and convolutional operations, which enables encoding all candidate NAS operations into an MSA layer in a single-path framework. We then cast the search problem as finding the subset of MSA parameters, thereby significantly reducing the search cost.
- Based on the single-path scheme, we propose SPViT that automatically prunes the costly and global MSA operations into the lightweight and local convolutional operations as well as searching for fine-grained MLP expansion ratios under desired efficiency constraints.
- Extensive experiments on CIFAR-100 [30] and ImageNet-1k [52] show that SPViT is able to significantly reduce the search cost while performing favorably against the baseline methods. We also make several interesting observations on the searched architectures revealing the architecture preferences for ViTs.

2. Related Work

Pruning Transformers. Transformers usually have expensive computational cost. To alleviate such cost, model pruning is one of the dominant approaches to derive efficient Transformers. Prior arts for pruning Transformers can be roughly categorized into module pruning and token pruning. The methods in the former category [62, 67] prune the insignificant Transformer modules, *e.g.*, heads in the MSA layers [3, 43], channels for the linear projections [32, 42] and weight neurons [12, 49]. Recently, Zhu *et al*. [67] propose to dynamically identify and prune the less-important token feature dimensions. Chen *et al*. [11] propose a prune-and-grow approach for Transformer parameters under the lottery ticket hypothesis [19]. The latter category focuses on pruning less-important tokens [46, 51], *e.g.*, DynamicViT [51] hierarchically prunes redundant tokens and achieves promising FLOPs reduction in the classification task. However, with some tokens eliminated, it is challenging for applying token pruning methods to the dense prediction tasks, *e.g.*, segmentation.

Our work aims at pruning the network modules, but in contrast to the previous work, we focus on pruning the superfluous global correlations in MSA layers. Instead of adding convolutions and MSA independently into the search space [60] in a multi-path manner, we consider the intrinsic relationships between the two types of operations and prune MSA layers into convolutional layers, which allows us to quickly deploy efficient Transformer models with low search cost. Note that our method is orthogonal to pruning with lottery ticket hypothesis [11, 19], which however is beyond the scope of this paper.

Convolutional vs. self-attention layers. Our work is motivated by the recent studies exploring the properties of convolutional and self-attention layers. For instance, convolutional layers show a strong capability for extracting local texture cues with convolutional inductive bias [5, 20]. On the other hand, self-attention layers tend to emphasize object shapes by modeling global correlations [8, 45].

To enjoy merits from both operations, one line of work treats convolutional and self-attention layers independently and forms hybrid models, *e.g.*, inserting convolutional layers in ViTs [37, 61] or stacking self-attentions on top of CNNs [7, 22]. Another line of work explores the intrinsic relationships between the two types of operations [13, 14]. Among them, pioneer work [14] shows that self-attention layers with the learnable quadratic position encoding and an abundant number of heads *can express any convolutional layers*. Following [14], ConViT [16] includes soft inductive bias as partial of attention scores in self-attention layers and Transformed CNN [15] learns to cast pre-trained convolutional layers into self-attention layers. Our approach also falls into the scope of the latter line of work. In contrast to the previous work, in light of the strong expressive

power of MSA layers, we propose a novel weight-sharing scheme showing that a subset of MSA operation parameters can express convolutional operations. The weight-sharing scheme enables us to encode all candidate NAS operations into MSA layers in a single-path framework to reduce the search cost. We further devise SPViT to automatically and efficiently prune MSA operations into lightweight convolutional operations, while simultaneously enjoying the beneficial effect of the locality.

3. Method

We start by introducing the proposed weight-sharing scheme between MSA and convolutional operations in Section 3.1. Then, we elaborate on our single-path vision transformer pruning approach that enjoys the benefits from the weight-sharing scheme in Section 3.2.

3.1. Weight-sharing between MSA and convolutional operations

The weight-sharing scheme refers to sharing a subset of parameters among different operations. The efficacy of the scheme has been demonstrated by a breadth of NAS approaches [6, 24, 53], *e.g.*, sharing weights among the candidate convolutional operations largely reduces the search cost [53]. In this paper, we introduce a novel weight-sharing scheme between MSA and convolutional operations.

Revisiting convolutional and MSA layers. Convolutional layers are the basic building blocks for CNNs. Let $\boldsymbol{X} \in \mathbb{R}^{n_w \times n_e \times c_{in}}$ and $\boldsymbol{W}^k \in \mathbb{R}^{k \times k \times c_{in} \times c_{out}}$ be the input features and the convolutional kernel with kernel size k, where n_w, n_e, c_{in} and c_{out} are the spatial width, height, input dimensions, and output dimensions, respectively. Standard convolutional layers aggregate the features within the local receptive field, which is defined in set $\mathbb{\Delta}_k := [-\lfloor \frac{k}{2} \rfloor, ..., \lfloor \frac{k}{2} \rfloor] \times [-\lfloor \frac{k}{2} \rfloor, ..., \lfloor \frac{k}{2} \rfloor]$. Formally, let $\mathcal{S} := [1, ..., n_w] \times [1, ..., n_e]$ be the index set for the feature width and height, the output for a standard convolutional layer at position $(i, j) \in \mathcal{S}$ can be expressed as

$$\mathrm{Conv}_k(\boldsymbol{X})_{i,j,:} := \sum_{(\delta_1,\delta_2)\in\mathbb{\Delta}_k} \boldsymbol{X}_{i+\delta_1,j+\delta_2,:} \boldsymbol{W}^k_{\delta_1,\delta_2,:,:}. \quad (1)$$

Transformers take MSA layers as the main building blocks. However, instead of aggregating only neighboring features, MSA layers have larger receptive fields that cover the entire sequence. Considering the same input $\boldsymbol{X}$ as in Eq. (1) as a set of c_{in} dimensional embeddings and letting $(l, m) \in \mathcal{S}$ be the index for any key feature embedding, the output for an MSA layer at position (i, j) can be defined as

$$\mathrm{MSA}(\boldsymbol{X})_{i,j,:} = \sum_{h\in[n_h]} \sum_{(l,m)\in\mathcal{S}} \mathrm{Att}(\boldsymbol{X})^h_{(i,j),(l,m)} \boldsymbol{V}_{l,m,h,:} \boldsymbol{W}^{\mathrm{o}}_{:,h,:}, \quad (2)$$

where $\mathrm{Att}(\boldsymbol{X})^h = \mathrm{Softmax}\left(\boldsymbol{Q}_{:,:,h,:} \boldsymbol{K}^T_{:,:,h,:} / \sqrt{c_h}\right)$, $\boldsymbol{Q} := \boldsymbol{X}\boldsymbol{W}^{\mathrm{qry}}, \boldsymbol{K} := \boldsymbol{X}\boldsymbol{W}^{\mathrm{key}}, \boldsymbol{V} := \boldsymbol{X}\boldsymbol{W}^{\mathrm{val}}$ and n_h denotes the number of heads in MSA layers. Given the output dimension for the h-th head $c_h = c_{in}/n_h$, we define $\boldsymbol{W}^{\mathrm{val}}$, $\boldsymbol{W}^{\mathrm{qry}}$ and $\boldsymbol{W}^{\mathrm{key}} \in \mathbb{R}^{c_{in} \times n_h \times c_h}$ as the corresponding learnable value, query and key linear projections. Therefore, $\mathrm{Att}(\boldsymbol{X})^h \in \mathbb{R}^{(n_w n_e)\times(n_w n_e)}$ denotes the attention map for the h-th head, and with the index of (i, j) and (l, m), $\mathrm{Att}(\boldsymbol{X})^h_{(i,j),(l,m)}$ becomes a scalar. With another learnable linear projection $\boldsymbol{W}^{\mathrm{o}} \in \mathbb{R}^{c_h \times n_h \times c_{in}}$, $\mathrm{MSA}(\boldsymbol{X})$ keeps the original dimension as $\boldsymbol{X}$. For simplicity, we omit positional encodings [57] in MSA layers and all the learnable bias terms accompanied with the learnable projections in both convolutional and MSA layers.

The computational complexity for MSA layers is $\mathcal{O}(n_e n_w c_{in}^2 + (n_e n_w)^2 c_{in})$ [48]. In this case, when modeling high-resolution feature maps that $n_e n_w \gg c_{in}$, the second quadratic term for modeling global correlations dominates the computation. Whilst, in standard convolutional layers outputting the features maps with the same width and height, the computational complexity for convolutional layers remains as $\mathcal{O}(k^2 n_e n_w c_{in} c_{out})$.

The weight-sharing scheme. In this section, we demonstrate how a subset of MSA operation parameters can express bottleneck convolutional operations [25] when $c_{out} = c_{in}$. We reveal the shared parameters for the two types of operations by gradually eliminating the parameters exclusive to the MSA operation.

First, since convolutional operations only process features within the local regions, we fix the attention score for the non-local correlations to 0 and the local correlations to 1 for each embedding when expressing convolutional operations with MSA parameters. Formally

$$\mathrm{Att}(\boldsymbol{X})^h_{(i,j),(l,m)} = \begin{cases} 1 & \text{if } (i-l, j-m) \in \mathbb{\Delta}_k \\ 0 & \text{otherwise} \end{cases}. \quad (3)$$

By substituting back into Eq. (2), we have

$$\mathrm{MSA}(\boldsymbol{X})^-_{i,j,:} = \sum_{h\in[n_h]} \sum_{(\delta_1,\delta_2)\in\mathbb{\Delta}_k} \boldsymbol{V}_{i+\delta_1,j+\delta_2,h,:} \boldsymbol{W}^{\mathrm{o}}_{:,h,:}. \quad (4)$$

Here we use $\mathrm{MSA}(\boldsymbol{X})^-$ to represent outputs with the locality restriction in Eq. (3). In Eq. (4), we force the heads of an MSA operation to attend to the positions within the local window of size $k \times k$ centered at (i, j).

Next, we show that we can further profile MSA operation parameters into a convolutional kernel. In analogous to [14], one way is defining a bijective mapping $\boldsymbol{f} : [n'_h] \to \mathbb{\Delta}_k$ that assigns heads in a selected head subset $[n'_h]$ to specific positions within the local windows that $[n'_h] \subseteq [n_h]$.

We can then substitute h with $\boldsymbol{f}(\delta_1, \delta_2)$ in Eq. (4), *i.e.*,

$$\mathrm{MSA}(\boldsymbol{X})^{-}_{i,j,:} = \sum_{(\delta_1,\delta_2)\in\boldsymbol{\Delta}_k} \boldsymbol{V}_{i+\delta_1,j+\delta_2,\boldsymbol{f}(\delta_1,\delta_2),:} \boldsymbol{W}^{\mathrm{o}}_{:,\boldsymbol{f}(\delta_1,\delta_2),:}. \tag{5}$$

Here in Eq. (5), each head attend to a certain position within the local window. Nevertheless, many ViT variants have $n_h < 9$, *e.g.* DeiT-Ti and DeiT-S [54], which would restrict the weight-sharing scheme to only the large architectures. Also, in MSA layers, different heads tend to attend to different areas as visualized in [14, 47], hence, it is non-trivial to define the subset $[n'_h]$ and mapping $\boldsymbol{f}$ that identify the most suitable heads for the local positions. To allow the weight-sharing scheme to be applicable for generic ViT architectures as well as easing the difficulty for defining $[n'_h]$ and $\boldsymbol{f}$, we employ learnable parameters $\boldsymbol{z} \in \mathbb{R}^{n_h\times k\times k}$ followed by a softmax function $\sigma(\cdot)$ to first ensemble the MSA heads into the convolutional kernel positions:

$$\mathrm{MSA}(\boldsymbol{X})^{-}_{i,j,:} = \sum_{h\in[n_h]} \sum_{(\delta_1,\delta_2)\in\boldsymbol{\Delta}_k} \sigma(\mathbf{z})_{h,\delta_1,\delta_2} \boldsymbol{V}_{i+\delta_1,j+\delta_2,h,:} \boldsymbol{W}^{\mathrm{o}}_{:,h,:}, \tag{6}$$

where $\sigma(\mathbf{z})_{h,\delta_1,\delta_2} = e^{\mathbf{z}_{h,\delta_1,\delta_2}} / \sum_{k=1}^{n_h} e^{\mathbf{z}_{k,\delta_1,\delta_2}}$. In this case, the ensembles of MSA heads with scaling factor $\sigma(\mathbf{z})_{h,\delta_1,\delta_2}$ learn to attend to pre-defined positions within the local $k\times k$ windows. We make sure the ensembles of heads remain the original scale by employing the softmax function on $\boldsymbol{z}$.

Next, according to [14], it is possible to profile $\boldsymbol{W}^{\mathrm{val}}_{:,h,:}\boldsymbol{W}^{\mathrm{o}}_{:,h,:}$ into a convolutional kernel from Eq. (6) by substituting $\boldsymbol{V}$ with $\boldsymbol{X}\boldsymbol{W}^{\mathrm{val}}$ and applying the associative law:

$$\widehat{\mathrm{Conv}}_k(\boldsymbol{X})_{i,j,:} = \sum_{(\delta_1,\delta_2)\in\boldsymbol{\Delta}_k} \boldsymbol{X}_{i+\delta_1,j+\delta_2,:} \widehat{\boldsymbol{W}}^{k}_{\delta_1,\delta_2,:,:}, \tag{7}$$

where $\widehat{\boldsymbol{W}}^{k}_{\delta_1,\delta_2,:,:} := \sum_{h\in[n_h]} \sigma(\mathbf{z})_{h,\delta_1,\delta_2} \left(\boldsymbol{W}^{\mathrm{val}}_{:,h,:}\boldsymbol{W}^{\mathrm{o}}_{:,h,:}\right)$.

Instead, for the complexity consideration, we choose to form a bottleneck convolution operation [25] and modify Eq. (6) as

$$\begin{aligned}
&\mathrm{BConv}_k(\boldsymbol{X})_{i,j,:} \\
=&\Big(\sum_{h\in[n_h]} \sum_{(\delta_1,\delta_2)\in\boldsymbol{\Delta}_k} \sigma(\mathbf{z})_{h,\delta_1,\delta_2} \boldsymbol{V}_{i+\delta_1,j+\delta_2,h,:}\Big)\overline{\boldsymbol{W}^{\mathrm{o}}} &(8)\\
=&\Big(\sum_{(\delta_1,\delta_2)\in\boldsymbol{\Delta}_k} \boldsymbol{X}_{i+\delta_1,j+\delta_2,:} \boldsymbol{W}^{k}_{\delta_1,\delta_2,:,:}\Big)\overline{\boldsymbol{W}^{\mathrm{o}}} &(9)\\
\Rightarrow&\mathrm{Conv}_k(\boldsymbol{X})_{i,j,:}\overline{\boldsymbol{W}^{\mathrm{o}}}, &(10)
\end{aligned}$$

where

$$\begin{cases} \boldsymbol{W}^{k}_{\delta_1,\delta_2,:,:} := \sum_{h\in[n_h]} \sigma(\mathbf{z})_{h,\delta_1,\delta_2} \boldsymbol{W}^{\mathrm{val}}_{:,h,:} \\ \overline{\boldsymbol{W}^{\mathrm{o}}} := \sum_{h\in[n_h]} \sum_{\delta_1,\delta_2\in\boldsymbol{\Delta}_k} \sigma(\mathbf{z})_{h,\delta_1,\delta_2} \boldsymbol{W}^{\mathrm{o}}_{:,h,:} \end{cases}. \tag{11}$$

Note that in this form, the convolutional operation output dimension $c_{out} = c_h$ and we project $\mathrm{Conv}_k(\boldsymbol{X}) \in \mathbb{R}^{n_w\times n_e\times c_h}$ back to $\mathbb{R}^{n_w\times n_e\times c_{in}}$ via $\overline{\boldsymbol{W}^{\mathrm{o}}}$. One can easily find that both $\mathrm{Rank}(\widehat{\mathrm{Conv}}_k(\boldsymbol{X}))$ in Eq. (7) and $\mathrm{Rank}(\mathrm{Conv}_k(\boldsymbol{X})\overline{\boldsymbol{W}^{\mathrm{o}}})$ in Eq. (10) are not greater than c_h, the dimension of heads. However, the computational complexity for the former is $\mathcal{O}(k^2 n_w n_e c_{in}^2)$ and for the latter is $\mathcal{O}(k^2 n_w n_e c_{in} c_h)$. Since $c_h \leq c_{in}$, we choose to employ the bottleneck convolutions for better efficiency.

We further add batch normalization (BN) [44] and ReLU non-linearity [29] and get the bottleneck convolutional operations for kernel size k expressed by $\boldsymbol{W}^{\mathrm{val}}$ and $\boldsymbol{W}^{\mathrm{o}}$:

$$\mathrm{BConv}_k(\boldsymbol{X})_{i,j,:} = \mathrm{BN}(\mathrm{ReLU}(\mathrm{Conv}_k(\boldsymbol{X})_{i,j,:}))\overline{\boldsymbol{W}_{\mathrm{o}}}. \tag{12}$$

Remark. The proposed weight-sharing scheme encourages the attention heads of an MSA operation to not only model the global regions, but also learn ensembles to process the local region features at the same time. The intuition for this approach mainly comes from two aspects. Firstly, the local regions are indeed part of the global regions, and thus the ensemble of MSA heads naturally has the capacity for handling local regions. Secondly, since some attention heads in ViT encoders often attend to the local regions around the query pixels [14, 50], expressing the behaviors for convolutional operations with these heads is likely to get us reasonable outputs. However, considering the optimal places to add locality might vary for different ViT models, we thus introduce SPViT that automatically learns to choose the optimal operations in Section 3.2.

Note that the proposed weight-sharing scheme has wide applications on ViT models without the conditions of using the learnable quadratic positional encoding and $n_h \geq k^2$ as required in [14]. The reason is that [14] learns the attention head subset $[n'_h]$ and the bijective mapping $\boldsymbol{f}$ in Eq. (5) between heads and the convolutional kernel positions with the learnable quadratic positional encoding, while we simplify this process by forming the convolutional kernels as ensembles of attention heads with $\sigma(\boldsymbol{z})$.

3.2. Single-path vision transformer pruning

Built upon the weight-sharing scheme, we further propose SPViT, to derive the architectures with optimal accuracy-efficiency trade-off by automatically pruning some MSA layers into convolutional layers. In addition, we also search for fine-grained MLP expansion ratios controlling the number of hidden dimensions in FFN layers. Benefiting from the weight-sharing scheme between MSA and convolutional operations, the search cost is largely reduced.

3.2.1 Search formulation

Searching for MSA and convolutional layers. We first search for the optimal places to prune MSA layers into convolutional layers in ViTs. Specifically, in each ViT block, we replace each original MSA layer with a Unified MSA

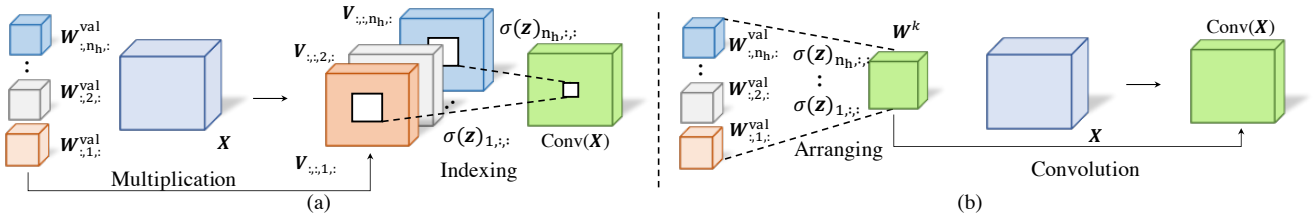

Figure 2. The process for deriving convolutional operation outputs during searching and fine-tuning. (a) During searching, as in Eq. (8), we get convolutional operation outputs by indexing and scaling $\boldsymbol{V}$, which also takes part in MSA calculations. In this case, the computations for convolutional candidate operations are largely saved. (b) During fine-tuning, if a convolutional operation is chosen, we first profile the scaled $\boldsymbol{W}^{\text{val}}$ into a convolutional kernel $\boldsymbol{W}^k$, and then directly perform the convolutional operation by $\boldsymbol{X} * \boldsymbol{W}^k$.

(UMSA) layer that gets the outputs for the MSA and convolutional operations simultaneously via the weight-sharing scheme described in Section 3.1. Then, we define a series of learnable binary gates corresponding to the operations to encode the architecture configuration.

To this end, in UMSA layers, when performing MSA operations, we can directly get the convolution output for pre-defined kernels $k \in \mathcal{K}$ by indexing and scaling $\boldsymbol{V}$ according to Eq. (8), without the need to keep each candidate convolutional operation as a separate path. The process is illustrated in Figure 2 (a). We then get the bottleneck convolutional operation outputs $\text{BConv}_k(\boldsymbol{X})$ with Eq. (12). In this way, we derive the outputs for both bottleneck convolutions and MSA with low computational cost. By the order of computational complexity from low to high, we can formulate the bottleneck convolutions and MSA into an ordered set $\mathcal{P}$, where $|\mathcal{P}| = |\mathcal{K}| + 1$.

Next, following [36, 40], we define a series of binary gates to encode the operation choices. For any operation with index $p \in [|\mathcal{P}|]$, gate g_p is sampled from a Bernoulli distribution with a learnable parameter θ_p which can be automatically optimized by gradients. Specifically, we use $\text{Sigmoid}(\theta_p)$ to determine the likelihood for choosing the p-th operation:

$$g_p \sim \text{Ber}(\text{Sigmoid}(\theta_p)). \tag{13}$$

For the efficiency consideration, we choose only one operation per layer. In this way, we enforce each UMSA layer to have at most one open gate by replacing g with $\hat{g}$:

$$\hat{g_p} = g_p \prod_{q=p+1}^{|\mathcal{P}|} (1 - g_q)\,. \tag{14}$$

Eq. (14) essentially states that once a more complex gate is selected ($g_q = 1$), all less complex gates will be closed. The output for the UMSA layer hence can be expressed as

$$O_{\text{UMSA}}(\boldsymbol{X}) = \sum_{p=1}^{|\mathcal{P}|} \hat{g_p} \cdot O_p(\boldsymbol{X}), \tag{15}$$

where $O_p(\boldsymbol{X})$ is the output for the p-th operation. Each UMSA layer is then followed by a residual connection and layer normalization [1] as ViTs do. The skip-connection operation is adopted when all gates are closed. Hence, our UMSA search space includes three types of candidate operations: skip-connections, bottleneck convolutions with different candidate kernel sizes, and MSA.

Searching for fine-grained MLP expansion ratios. MLP expansion ratios control the number of hidden dimensions for the FFN layers. Previous work [9, 10, 38] define the search space for the MLP expansion ratios as a list of coarse-grained values, *e.g.*, $[3, 3.5, 4]$ in [10]. Instead, given the MLP expansion ratio α defined by the pre-trained model, we make the first attempt to search for fine-grained MLP expansion ratios α' that $0 \leq \alpha' \leq \alpha$ for each FFN layer via a simple but effective Unified FFN (UFFN) layer. Given input features $\boldsymbol{X}$, the output for FFN layers can be expressed as

$$O_{\text{FFN}}(\boldsymbol{X}) = \sum_{t \in [\alpha c_{in}]} O_t(\boldsymbol{X}), \tag{16}$$

where $O_t(\boldsymbol{X}) = \text{GeLU}\left(\boldsymbol{X}\boldsymbol{W}^{\text{fc1}}_{:,t}\right)\boldsymbol{W}^{\text{fc2}}_{t,:}$. Here we use α and αc_{in} to denote the pre-defined MLP expansion ratio and the hidden dimension for FFN layers. $\boldsymbol{W}^{\text{fc1}} \in \mathbb{R}^{c_{in} \times \alpha c_{in}}$ and $\boldsymbol{W}^{\text{fc2}} \in \mathbb{R}^{\alpha c_{in} \times c_{in}}$ are learnable fully-connected projections where the former maps the channel dimension of $\boldsymbol{X}$ to αc_{in} and the latter projects the channel dimension back to c_{in}. GeLU [27] activation is added between the two learnable fully-connected projections.

Same as Eq. (13), we search for fine-grained MLP expansion ratios by encoding the FFN hidden dimension configurations with binary gates. By applying the binary gates to each hidden dimension during the search, we define the UFFN layer output as

$$O_{\text{UFFN}}(\boldsymbol{X}) = \sum_{t \in [\alpha c_{in}]} g_t \cdot O_t(\boldsymbol{X}). \tag{17}$$

This allows us to prune out unimportant hidden dimensions.

Searching objective. To get the architectures with desired efficiency constraints, we optimize the network with an aux-

iliary computational complexity loss. Specifically, we define a look-up table containing the computational complexities for the candidate operations and modules. The computational cost $\mathcal{L}_{\text{comp}}$ is defined as

$$\mathcal{L}_{\text{comp}} = (\mathrm{F}(\boldsymbol{X}) - \hat{\mathrm{F}}(\boldsymbol{X}))^2, \tag{18}$$

where $\mathrm{F}(\boldsymbol{X})$ is the current network computational complexity and $\hat{\mathrm{F}}(X)$ is the target one. Finally, the overall searching objective is defined as $\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda\mathcal{L}_{\text{comp}}$, where λ is the trade-off hyper-parameter balancing $\mathcal{L}_{\text{comp}}$ and the cross-entropy loss $\mathcal{L}_{\text{CE}}$.

3.2.2 Fine-tuning

During fine-tuning, we make the previous stochastic binary gates in Section 3.2.1 deterministic, *i.e.*,

$$g'_p = \mathbb{1}(\text{Sigmoid}(\theta_p) \geq 0.5). \tag{19}$$

By applying Eq. (14), we obtain $\hat{g_p}'$ and select only at most one operation in UMSA layers during fine-tuning. The output can be expressed as

$$O'_{\text{UMSA}}(\boldsymbol{X}) = \begin{cases} \text{MSA}(\boldsymbol{X}) & \text{if } \hat{g_p}' = 1 \quad \text{and} \quad p = |\mathcal{P}| \\ \text{BConv}'_k(\boldsymbol{X}) & \text{if } \hat{g_p}' = 1 \quad \text{and} \quad p < |\mathcal{P}| \\ 0 & \text{otherwise} \end{cases}. \tag{20}$$

If the selected operation is bottleneck convolution, we can profile a well-trained $\boldsymbol{W}^{\text{val}}$ into a convolutional kernel to seek for efficient hardware implementation as described in Eq. (10) and depicted in Figure 2 (b). Combining with Eq. (12), we can define the bottleneck convolutional layer during fine-tuning as

$$\text{BConv}'_k(\boldsymbol{X}) = \text{BN}(\text{ReLU}(\boldsymbol{X} * \boldsymbol{W}^k)\overline{\boldsymbol{W}_{\text{o}}}, \tag{21}$$

where $\boldsymbol{W}^k = \text{Arrange}(\boldsymbol{W}^k_{1,1,:,:}, ..., \boldsymbol{W}^k_{k,k,:,:})$. Similarly, for UFFN layers, we can automatically find a subset of hidden dimensions $\mathcal{T} \subseteq [\alpha c_{in}]$ indicating the selected hidden dimensions. Therefore, α' is $|\mathcal{T}|/c_{in}$ and the UFFN layer outputs during fine-tuning can be derived by $O'_{\text{UFFN}}(X) = \sum_{t \in \mathcal{T}} O_t(\boldsymbol{X})$. During fine-tuning, the objective is set to $\mathcal{L}_{\text{CE}}$.

4. Experiment

In this section, we validate the effectiveness of our SPViT by pruning different ViT models. We conduct experiments on two series of ViT models: DeiT [54] and Swin [41], which are representative standard ViT and hierarchical ViT [48, 58] models. In the following, we first compare with the baseline pruning methods in Section 4.1, then analyze some observed patterns when varying the expected computational complexity in Section 4.2, and finally ablate some important components of SPViT in Section 4.3.

Models	Top-1 Acc. (%)	Top-5 Acc. (%)	Params (M)	FLOPs (G)
DeiT-Ti [54]	72.2	91.1	5.7	1.3
SBP-DeiT-Ti* [2, 43]	68.6	-	4.2	1.0
SPViT-DeiT-Ti	**70.7**	**90.3**	4.9	1.0
DeiT-S [54]	79.9	95.0	22.1	4.6
SBP-DeiT-S* [2, 43]	77.7	-	14.6	3.2
SPViT-DeiT-S	**78.3**	**94.3**	16.4	3.3
DeiT-B [54]	81.8	95.6	86.4	17.5
VPT-DeiT-B [67]	81.3	95.3	67.3	13.8
SBP-DeiT-B [2, 43]	80.1	-	56.8	11.7
SPViT-DeiT-B	**81.6**	**95.5**	62.3	11.7

Table 1. ImageNet-1k results for pruning DeiT [54]. * indicates pruning from scratch.

Models	Top-1 Acc. (%)	Top-5 Acc. (%)	Params (M)	FLOPs (G)
Swin-Ti [41]	81.2	95.5	28.3	4.5
STEP-Swin-Ti [35]	77.2	93.6	23.6	3.5
SPViT-Swin-Ti	**80.1**	**95.0**	25.8	3.4
Swin-S [41]	83.2	96.2	49.6	8.7
STEP-Swin-S [35]	79.6	94.7	36.9	6.3
SPViT-Swin-S	**81.8**	**95.9**	40.1	6.3
Swin-B [41]	83.5	96.5	87.8	15.4
STEP-Swin-B [35]	80.6	95.3	69.3	12.1
SPViT-Swin-B	**82.8**	**96.2**	72.2	12.1

Table 2. ImageNet-1k results for pruning Swin [41].

Experimental setup. We perform both searching and fine-tuning from the pre-trained models. Unless specified, the training details for our SPViT align with those introduced in DeiT [54] and Swin [41]. During the search, we set the initial learning rates for the gate parameters θ and other network parameters as 1×10^{-3} and 1×10^{-4}, respectively. The initialized value for the gate parameters θ is set to 1.5 to initialize a high probability for choosing MSA operations. In UMSA layers, we include the most popular 1×1 and 3×3 bottleneck convolutions in our search space. The network architectures converge at around 10 epochs and 50 epochs for models trained on ImageNet-1k [52] and CIFAR-100 [30], respectively (ImageNet-1k has a larger training set thereby converging faster), and then, we follow [39] and start fine-tuning for 130 epochs on ImageNet-1k [52] and 300 epochs on CIFAR-100 [30] from the searched architectures with a learning rate of 1×10^{-4}. More implementation details can be found in supplementary materials.

Compared methods. We include two structured pruning baselines for comparison. The first one is Sensitivity-Based Pruning (SBP) that iteratively prunes the unimportant attention heads for MSA layers and channels for FFN layers according to their sensitivity to the loss as illustrated in [2,43]. Here we borrow the implementations from [11]. The second one is Straight-Through Estimator Pruning (STEP) that learns the importance scores for attention heads of MSA

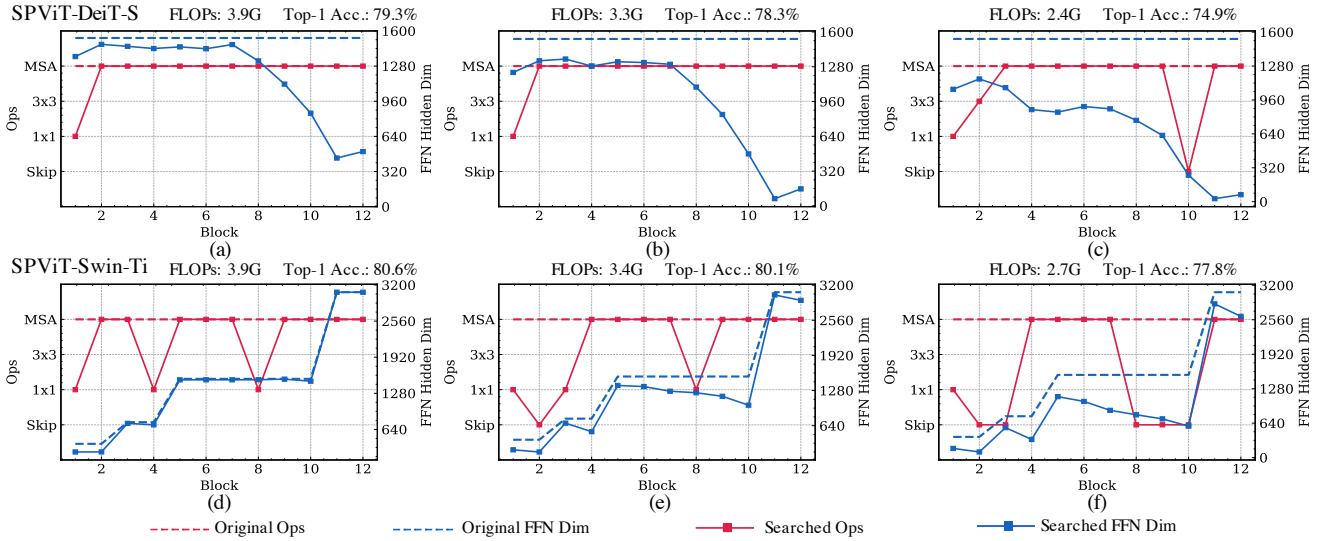

Figure 3. SPViT searched architectures under different target FLOPs on ImageNet-1k. (a)-(c) are architectures searched by SPViT-DeiT-S and (d)-(f) are architectures searched by SPViT-Swin-Ti. Dashed lines represent the original operations and the number of hidden dimensions for FFN layers before pruning. Solid lines represent the two types of architecture configurations searched by SPViT.

layers and hidden dimensions of FFN layers with the learnable gate parameters optimized by the Straight-Through Estimator [4]. The top-k attention heads and hidden dimensions with the highest importance scores are kept. The implementations for STEP are borrowed from [35]. Aside from the baseline methods, we also compare our method with the previous pruning method VPT [67]. We name all these methods as "(Pruning method)-(Dense model)", *e.g.*, SPViT-DeiT-Ti and SPViT-Swin-S denote our method that prunes DeiT-Ti and Swin-S models, respectively.

4.1. Main Results

We investigate the effectiveness of our method by comparing SPViT with baseline methods in Tables 1 and 2. In Table 1, we can observe that SPViT prunes the dense DeiT models into compact ones saving more than 20% FLOPs with competitive Top-1 and Top-5 accuracy. For instance, our SPViT-DeiT-B achieves 33.5% FLOPs saving while only incurring 0.2% Top-1 accuracy performance drop. We also notice that our SPViT consistently outperforms the baseline pruning methods in terms of both Top-1 and Top-5 accuracy. Note that although the SBP method [2, 43] prunes DeiT models from scratch while our SPViT prunes from the pre-trained weights, SBP has much longer training epochs (600 epochs) than our SPViT (130 epochs).

When pruning the relatively compact hierarchical ViT models, we can observe in Table 2 that SPViT also achieves remarkable performance when pruning more than 20% FLOPs. For example, SPViT-Swin-B achieves 3.3G FLOPs saving with only 0.7% performance drop on Top-1 accuracy, which is 2.2% higher compared to STEP-Swin-B.

4.2. Observations on searched architectures

Our SPViT can automatically find well-performed architectures under desired efficiency constraints during the search. Here, we show empirical observations on the searched architectures using SPViT on standard and hierarchical ViTs when gradually decreasing the target FLOPs. In Figure 3, we visualize the searched architectures for SPViT-DeiT-S under 3.9G, 3.2G and 2.4G FLOPs with respectively 79.3%, 78.3% and 74.9% Top-1 accuracy and SPViT-Swin-Ti under 3.9G, 3.4G and 2.7G FLOPs with 80.6%, 80.1% and 77.8% Top-1 accuracy, respectively. More visualizations for the architectures searched by different SPViT variants can be found in Section **??**.

Locality is encouraged in shallow blocks. As depicted in Figure 3 (a)-(f), SPViT-DeiT-S and SPViT-Swin-Ti learn to prune the shallow MSA layers into bottleneck convolutional layers or skip-connections (in the first two blocks for SPViT-DeiT-S and the first three blocks for SPViT-Swin-Ti). This observation aligns with the previous work [14, 47] indicating that the shallow MSA layers contain more superfluous global correlations. It can also be seen that the last two blocks remain as the MSA layers for all models, demonstrating the importance of the global operations in the deep layers. More analyses on the attention map patterns of the pre-trained MSA layers can be found in the supplementary material.

Last few FFN layers have more redundancy in standard ViTs. For the pre-trained DeiT models, the MLP expansion ratios are set to be the same across all the blocks. As illustrated in Figure 3 (a)-(c), for the SPViT-DeiT-S settings,

Model	Top-1 Acc. (%)	Params (M)	FLOPs (G)	Search Cost (GPU hours)
SPViT-DeiT-Ti w/ MP	70.5	5.0	1.0	18.1
SPViT-DeiT-Ti	**70.7**	**4.9**	1.0	**8.6**
SPViT-DeiT-S w/ MP	78.0	16.5	3.3	28.9
SPViT-DeiT-S	**78.3**	**16.4**	3.3	**12.0**

Table 3. Effect of our single-path search on ImageNet-1k. "SPViT-Model w/MP" denotes our pruning method except changing single-path to multi-path.

Model	Top-1 Acc. (%)	Top-5 Acc. (%)	FLOPs (G)
Swin-Ti	77.9	95.1	4.5
Random Search	76.1±0.1	94.2±0.1	3.2
SPViT-Swin-Ti	**77.4±0.0**	**94.8±0.1**	**3.2**

Table 4. Comparison with random search on CIFAR-100. We run the experiments for 5 times and report the results with mean and standard deviation.

the networks prioritize pruning the hidden dimensions in the last few blocks. As a higher number of hidden dimensions in FFN layers indicates a higher capacity and vice versa [17, 31], we conjecture that the last few FFN layers need less capacity compared to the others since they handle less diverse attention maps [21, 55, 66] in standard ViTs.

Shallower FFN layers within each stage require higher capacity in hierarchical ViTs. Different from standard ViTs, the Swin models are with hierarchical architectures that separate the blocks into several stages. At the beginning of each stage, the input tokens are merged into higher embedding dimensions. As shown in Figure 3 (d)-(f), within the same stage, the shallower layers preserve more hidden dimensions. We conjecture that modeling the changes on the feature space caused by token merging requires a higher capacity for hierarchical ViTs.

4.3. Ablation studies

Single-path vs. multi-path search. We investigate the effectiveness of our single-path search by comparing SPViT with the multi-path counterpart in Table 3. Specifically, in the multi-path implementations, we randomly initialize the weights in candidate bottleneck convolution operations before the search and keep other components the same as the single-path version. We observe that under the same computational complexity, our single-path formulation achieves higher performance, fewer parameters, and lower search cost compared with the search in a multi-path manner. For example, SPViT-DeiT-S surpasses the multi-path counterpart by 0.3% Top-1 accuracy and achieves more than 50% search cost reduction.

Effect of the search strategy. To validate the effectiveness of our searching strategy, we compare our method with randomly searched architectures on CIFAR-100 in Table 4. Specifically, we randomly sample 10 architectures under our search space with computational complexity around the target FLOPs and then fine-tune these architectures. We compare SPViT with the architecture achieving the highest Top-1 accuracy. We observe that under similar FLOPs and parameters, our SPViT excels the random search counterpart by around 1.3% and 0.6% in terms of the Top-1 and Top-5 accuracy. The significant performance gain demonstrates the effectiveness of our searching strategy.

Model	FLOPs Saving (%)	MSA Saving (%)	FFN Saving (%)
SPViT-DeiT-B	33.1	34.5	30.8
SPViT-Swin-Ti	24.4	34.2	20.6
SPViT-Swin-S	27.2	21.3	32.1
SPViT-Swin-B	21.4	21.2	23.0

Table 5. Pruning proportions for individual components. The listed SPViT variants are the same ones as in Table 1 and Table 2.

Pruning proportions for MSA and FFN layers. We investigate the pruning proportions for MSA and FFN layers in Table 5. We can observe that the pruning proportions are different among the models, *e.g.*, SPViT-DeiT-B has a higher pruning ratio for MSA layers while SPViT-Swin-B has a higher pruning ratio for FFN layers. It is suggested that given target efficiency constraints, SPViT can flexibly customize the suitable pruning proportions for different dense models.

5. Conclusion and future work

In this paper, we have introduced a novel weight-sharing scheme between MSA and convolutional operations, which allows encoding convolutional operations with a subset of MSA parameters and optimizing the two types of operations simultaneously. Based on the weight-sharing scheme, we have proposed SPViT to reduce the computational cost of ViTs as well as introducing proper inductive bias automatically with low search cost. Specifically, SPViT can search whether to prune the MSA into convolutional layers in ViTs and search for optimal hidden dimensions of FFN layers under desired efficiency constraints. By applying SPViT to two popular ViT variants, we have made some meaningful observations towards the importance of ViT components.

Limitations and social impacts. One limitation of SPViT is that the granularity for pruning MSA into convolutional layers is too coarse, where pruning the former to the latter incurs a large loss of discriminative capability due to the huge parameter gap. Hence, we will explore adding head pruning into SPViT as future work. Another future direction is to explore extending the weight-sharing scheme to other convolutional operation variants such as depth-wise convolution [28] and group convolution [64], providing wider applications. For social impacts, although our pruning approach can prune the dense models into lightweight ones and save computational resources, the training process still leads to large carbon emissions, especially when deriving the pre-trained models.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 5

[2] Brian R Bartoldson, Ari S Morcos, Adrian Barbu, and Gordon Erlebacher. The generalization-stability tradeoff in neural network pruning. In *NeurIPS*, 2020. 6, 7

[3] Maximiliana Behnke and Kenneth Heafield. Losing heads in the lottery: Pruning transformer attention in neural machine translation. In *EMNLP*, pages 2664–2674, 2020. 2

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 7

[5] Wieland Brendel and Matthias Bethge. Approximating cnns with bag-of-local-features models works surprisingly well on imagenet. In *ICLR*, 2019. 2

[6] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020. 3

[7] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, pages 213–229, 2020. 1, 2

[8] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021. 2

[9] Boyu Chen, Peixia Li, Chuming Li, Baopu Li, Lei Bai, Chen Lin, Ming Sun, Junjie Yan, and Wanli Ouyang. Glit: Neural architecture search for global and local image transformer. In *ICCV*, pages 12–21, 2021. 1, 5

[10] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *ICCV*, pages 12270–12280, 2021. 5

[11] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. In *NeurIPS*, 2021. 1, 2, 6

[12] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. In *NeurIPS*, 2020. 2

[13] Xuanhong Chen, Hang Wang, and Bingbing Ni. X-volution: On the unification of convolution and self-attention. *arXiv preprint arXiv:2106.02253*, 2021. 2

[14] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. In *ICLR*, 2020. 2, 3, 4, 7

[15] Stéphane d'Ascoli, Levent Sagun, Giulio Biroli, and Ari Morcos. Transformed cnns: recasting pre-trained convolutional layers with self-attention. *arXiv preprint arXiv:2106.05795*, 2021. 2

[16] Stéphane d'Ascoli, Hugo Touvron, Matthew Leavitt, Ari Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *ICML*, 2021. 2

[17] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *ICML*, 2021. 8

[18] 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 *ICLR*, 2021. 1

[19] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019. 2

[20] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. In *ICLR*, 2019. 2

[21] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *ICML*, pages 3690–3699, 2020. 8

[22] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Herve Jegou, and Matthijs Douze. Levit: A vision transformer in convnet's clothing for faster inference. In *ICCV*, pages 12259–12269, October 2021. 2

[23] 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. 1

[24] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, pages 544–560, 2020. 3

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 3, 4

[26] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, pages 784–800, 2018. 1

[27] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 5

[28] 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. 8

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. 4

[30] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009. 2, 6, 12

[31] Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *NeurIPS*, 2019. 8

[32] Bingbing Li, Zhenglun Kong, Tianyun Zhang, Ji Li, Zhengang Li, Hang Liu, and Caiwen Ding. Efficient transformer-based large scale language representations using hardware-friendly block structured pruning. In *EMNLP*, 2020. 2

[33] Changlin Li, Tao Tang, Guangrun Wang, Jiefeng Peng, Bing Wang, Xiaodan Liang, and Xiaojun Chang. Bossnas: Exploring hybrid cnn-transformers with block-wisely self-supervised neural architecture search. In *ICCV*, 2021. 1

[34] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *CVPR*, pages 8607–8617, 2021. 1

[35] Jiaoda Li, Ryan Cotterell, and Mrinmaya Sachan. Differentiable subset pruning of transformer heads. *TACL*, 2021. 6, 7

[36] Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy Hospedales, Neil M Robertson, and Yongxin Yang. Differentiable automatic data augmentation. In *ECCV*, pages 580–595, 2020. 5

[37] Yawei Li, Kai Zhang, Jiezhang Cao, Radu Timofte, and Luc Van Gool. Localvit: Bringing locality to vision transformers. *arXiv preprint arXiv:2104.05707*, 2021. 1, 2

[38] Yi-Lun Liao, Sertac Karaman, and Vivienne Sze. Searching for efficient multi-stage vision transformers. *arXiv preprint arXiv:2109.00642*, 2021. 5

[39] Jing Liu, Bohan Zhuang, Peng Chen, Yong Guo, Chunhua Shen, Jianfei Cai, and Mingkui Tan. Lbs: Loss-aware bit sharing for automatic model compression. *arXiv preprint arXiv:2101.04935*, 2021. 6

[40] Jing Liu, Bohan Zhuang, Mingkui Tan, Xu Liu, Dinh Phung, Yuanqing Li, and Jianfei Cai. Elastic architecture search for diverse tasks with different resources. *arXiv preprint arXiv:2108.01224*, 2021. 5

[41] 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, 6

[42] Jiachen Mao, Huanrui Yang, Ang Li, Hai Li, and Yiran Chen. Tprune: Efficient transformer pruning for mobile devices. *TCPS*, 5(3):1–22, 2021. 2

[43] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *NeurIPS*, volume 32, 2019. 2, 6, 7

[44] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010. 4

[45] Muzammal Naseer, Kanchana Ranasinghe, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Intriguing properties of vision transformers. In *NeurIPS*, 2021. 2

[46] Bowen Pan, Yifan Jiang, Rameswar Panda, Zhangyang Wang, Rogerio Feris, and Aude Oliva. Ia-red^2: Interpretability-aware redundancy reduction for vision transformers. In *NeurIPS*, 2021. 2

[47] Zizheng Pan, Bohan Zhuang, Haoyu He, Jing Liu, and Jianfei Cai. Less is more: Pay less attention in vision transformers. *arXiv preprint arXiv:2105.14217*, 2021. 4, 7

[48] Zizheng Pan, Bohan Zhuang, Jing Liu, Haoyu He, and Jianfei Cai. Scalable visual transformers with hierarchical pooling. In *ICCV*, 2021. 1, 3, 6

[49] Sai Prasanna, Anna Rogers, and Anna Rumshisky. When bert plays the lottery, all tickets are winning. In *EMNLP*, 2020. 2

[50] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *arXiv preprint arXiv:2108.08810*, 2021. 2, 4

[51] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *NeurIPS*, 2021. 2

[52] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 2, 6, 12

[53] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *ECML PKDD*, 2020. 3

[54] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, pages 10347–10357, 2021. 1, 2, 4, 6

[55] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *ICCV*, 2021. 8

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017. 1

[57] 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*, pages 5998–6008, 2017. 3

[58] 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. 6

[59] Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross Girshick. Early convolutions help transformers see better. In *NeurIPS*, 2021. 1

[60] Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. Nas-bert: Task-agnostic and adaptive-size bert compression with neural architecture search. In *KDD*, 2021. 1, 2

[61] Yufei Xu, Qiming Zhang, Jing Zhang, and Dacheng Tao. Vitae: Vision transformer advanced by exploring intrinsic inductive bias. In *NeurIPS*, 2021. 1, 2

[62] Huanrui Yang, Hongxu Yin, Pavlo Molchanov, Hai Li, and Jan Kautz. Nvit: Vision transformer compression and parameter redistribution. *arXiv preprint arXiv:2110.04869*, 2021. 2

[63] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. In *NeurIPS*, 2019. 1

[64] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, pages 6848–6856, 2018. 8

[65] 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*, pages 6881–6890, 2021. 1

[66] Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, and Jiashi Feng. Deepvit: Towards deeper vision transformer. *arXiv preprint arXiv:2103.11886*, 2021. 8

[67] Mingjian Zhu, Kai Han, and Yehui Tang. Visual transformer pruning. In *KDDW*, 2021. 1, 2, 6, 7

Appendix

We organize the appendix as follows.

- In Section A, we provide more implementation details for our SPViT.
- In Section B, we show and analysis the architectures searched by large SPViT variants.
- In Section C, we analyse the attention probability patterns for DeiT-B pre-trained model.

A. More implementation details

For experiments on ImageNet-1k [52], we search and fine-tune our models with a total batch size of 1,024 on 8 NVIDIA V100 GPUs except that SPViT-DeiT-Ti is trained on 4 NVIDIA V100 GPUs. We also evaluate our models with the same hardware specifications. For experiments on CIFAR-100 [30], we search and fine-tune our models with a batch size of 84 on a single NVIDIA V100 GPU. We also set the learning rate warm-up epochs for all models to be 0.

B. Architectures searched by large SPViT variants

Here we show architectures searched by SPViT-DeiT-B and SPViT-Swin-B in Figure 4. We can see that in Figure 4 (a)-(f), the shallow MSA layers are pruned into bottleneck convolutional layers or skip-connections (in the first 3-4 blocks for SPViT-DeiT-B and the first three blocks for SPViT-Swin-B). Besides, in Figure 4 (a)-(c), the last few FFN layers have relatively lower remaining FFN hidden dimensions compared with the middle layers for SPViT-DeiT-B. Moreover, we can observe descending trends on FFN hidden dimensions within each stage for SPViT-Swin-B in Figure 4 (d)-(f). In a nutshell, the searched architectures for SPViT-DeiT-B and SPViT-Swin-B align with those for SPViT-DeiT-S and SPViT-Swin-Ti in the main paper showing that the observations are general for the ViT models.

C. Analyses on the attention probability patterns

To further support our observation that locality is encouraged in shallow blocks, we visualize the attention probabilities for DeiT-B pre-trained model in Figure 5. We can observe that the attention probabilities show clear different patterns between the shallow and deep blocks. In shallow blocks (Blocks 1 and 2), many MSA heads attend to the local regions around the query feature embedding, while nearly all heads in the deep blocks (Blocks 11 and 12) attend to the global regions. Therefore,

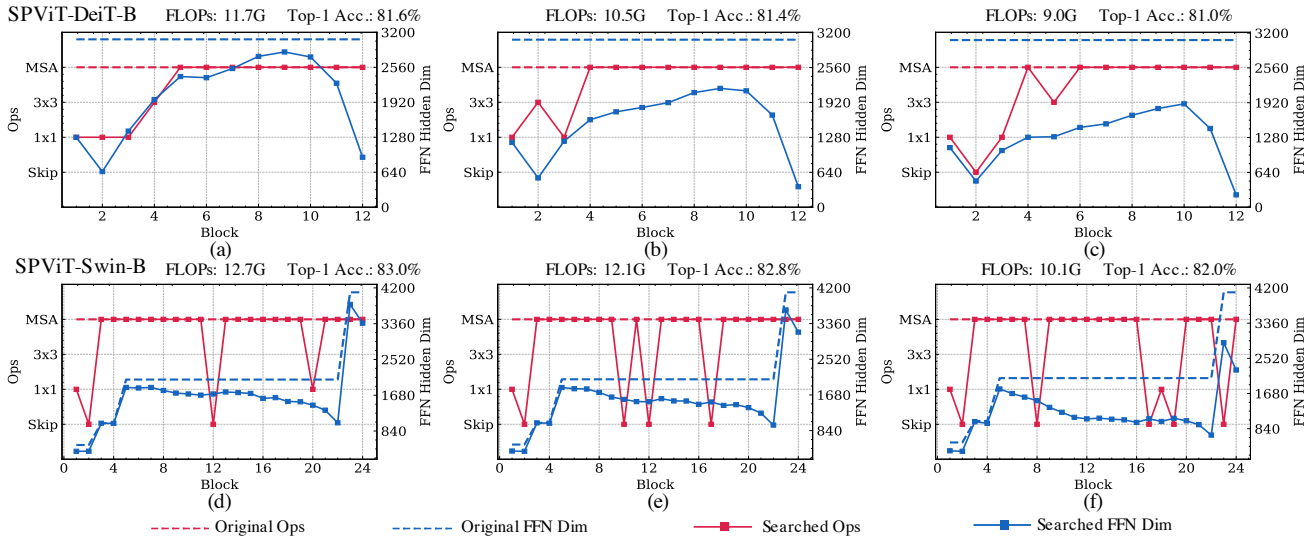

Figure 4. SPViT searched architectures under different target FLOPs on ImageNet-1k. (a)-(c) are architectures searched by SPViT-DeiT-B and (d)-(f) are architectures searched by SPViT-Swin-B. Dashed lines represent the original operations and the number of hidden dimensions for FFN layers before pruning. Solid lines represent the two types of architecture configurations searched by SPViT.

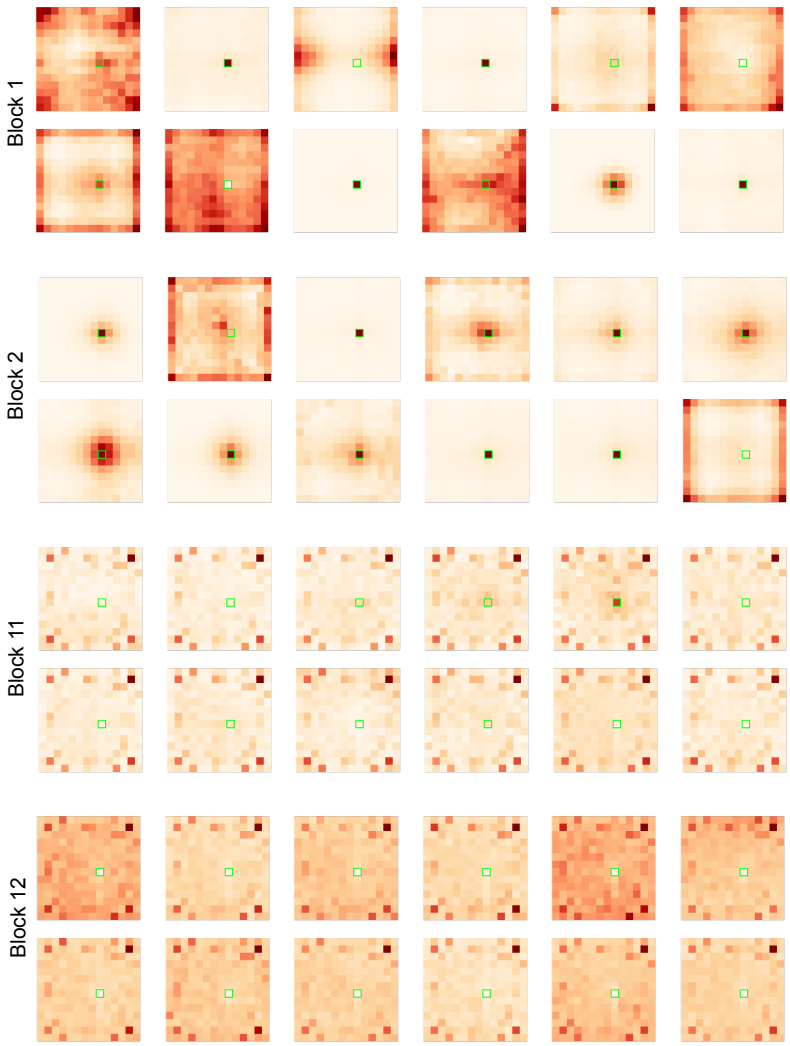

Figure 5. Attention probabilities of the pre-trained DeiT-B model averaged over 100 images. We visualize the attention maps for all twelve heads in the shallow blocks (Block 1 and 2) and deep blocks (Block 11 and 12). Each map shows the attention probabilities between a query feature embedding (*green square*) and the other feature embeddings. Darker color indicates higher attention probability and vice versa. Best viewed in color.

the attention probability patterns well justify that our SPViT encourages pruning the shallow MSA layers into convolutional layers and keeping the deep MSA layers unchanged.