Searching the Search Space of Vision Transformer

Minghao Chen, Kan Wu, Bolin Ni, Houwen Peng, Bei Liu, Jianlong Fu, Hongyang Chao, Haibin Ling

Stony Brook University, Sun Yat-sen University, Institute of Automation, CAS, Microsoft Research Asia

Abstract

Vision Transformer has shown great visual representation power in substantial vision tasks such as recognition and detection, and thus been attracting fast-growing efforts on manually designing more effective architectures. In this paper, we propose to use neural architecture search to automate this process, by searching not only the architecture but also the search space. The central idea is to gradually evolve different search dimensions guided by their E-T Error computed using a weight-sharing supernet. Moreover, we provide design guidelines of general vision transformers with extensive analysis according to the space searching process, which could promote the understanding of vision transformer. Remarkably, the searched models, named S3 (short for Searching the Search Space), from the searched space achieve superior performance to recently proposed models, such as Swin, DeiT and ViT, when evaluated on ImageNet. The effectiveness of S3 is also illustrated on object detection, semantic segmentation and visual question answering, demonstrating its generality to downstream vision and vision-language tasks. Code and models will be available at here.

1 Introduction

Vision transformer recently has drawn great attention in computer vision because of its high model capability and superior potentials in capturing long-range dependencies. Building on top of transformers [41], modern state-of-the-art models, such as ViT [10] and DeiT [40], are able to achieve competitive performance on image classification and downstream vision tasks [13, 20] compared to CNN models. There are fast-growing efforts on manually designing more effective architectures to further unveil the power of vision transformer [27, 53, 15].

Neural architecture search (NAS) as a powerful technique for automating the design of networks has shown its superiority to manual design [58, 39]. For NAS methods, search space is crucial because it determines the performance bounds of architectures during the search. It has been observed that the improvement of search space induced a favorable impact on the performance of many state-of-the-art models [2, 16, 50]. Researchers have devoted ample efforts to the design of CNN space [45, 12, 42]. However, limited attention is paid to Transformer counterparts, leading to the dilemma of search space design when finding more effective and efficient transformer architectures.

In this paper, we propose to Search the Search Space (S3) by automatically refining the main changeable dimensions of transformer architectures. In particular, we try to answer the following two key questions: 1) How to effectively and efficiently measure a specific search space? 2) How to automatically change a defective search space to a good one without human prior knowledge?

For the first question, we propose a new measurement called E-T Error for search space quality estimation and utilize a once-for-all supernet for effective and efficient computing. Specifically, the “E” part in the E-T error is similar to the error empirical distribution function used in [35] focusing on

*Equal contributions. Work done when Minghao, Kan, Bolin are interns of MSRA. †Corresponding author.
the overall quality, while the “T” part considers the quality of top-tier architectures in a search space. In contrast to [35] that trains each subnet for only a few epochs, we train a once-for-all supernet with AutoFormer [6] to obtain a large amount of well-trained subnets, providing a more reliable proxy for performance estimation.

For the second question, as shown in Fig. 1 we decompose the search space by multiple dimensions, including depth, embedding dimension, MLP ratio, window size, number of heads and $Q$-$K$-$V$ dimension, and progressively evolve each dimension to compose a better space. Particularly, we use linear parameterization to model the tendencies of different dimensions so as to guide the search.

In addition, we analyze the search process of search space and provide some interesting observations and design guidelines: 1) The third stage is the most important one and increasing its number of blocks leads to performance improvement. 2) Shallow layers should use a small window size while deep layers should use a large window size. 3) MLP ratio is better to be progressively increased along with network depth. 4) $Q$-$K$-$V$ dimension could be smaller than the embedding dimension without performance drop. We hope these observations will help both the manual design of vision transformer and the space design for automatic search.

In summary, we make the following contributions:

- We propose a new scheme, i.e. S3, to automate the design of search space for vision transformer. We also present a new pipeline for vision transformer search with minimal human interactions. Moreover, we provide analysis and guidelines on vision transformer architecture design, which might be helpful for future architecture search and design.

- The experiments on ImageNet verify the proposed automatic search space design method can improve the effectiveness of design space, thus boosting the performance of searched architectures. The discovered models achieve superior performance compared to the recent ViT [10] and Swin [27] transformer families under aligned settings. Moreover, the experiments on downstream vision and vision-language tasks, such as object detection, semantic segmentation and visual question answering, demonstrate the generality of the method.

## 2 Approach

### 2.1 Problem Formulation

Most of existing NAS methods can be formulated as a constrained optimization problem as:

$$
\alpha^*_A = \arg \min_{\alpha \in A} \mathcal{L}(W^*_\alpha; D_{val}) \quad s.t. \quad W^*_\alpha = \arg \min_{W_\alpha} \mathcal{L}(W_\alpha; D_{train}), \quad g(\alpha) < c,
$$

where $W_\alpha$ is the weights of the network architecture $\alpha$, $A$ is a predefined search space, $\mathcal{L}(\cdot)$ is the loss function, $D_{train}$ and $D_{val}$ represent train and validation datasets respectively, $g(\cdot)$ denotes a function calculating resource consumption of a model while $c$ denotes a given resource constraint.

Ideally, the search space $A$ is an infinite space $\Omega$ that contains all possible architectures. In practice, $A$ is usually a small subspace of $\Omega$ due to limited computation resource. In this paper, we break the
convention of searching in a fixed space by considering the architecture search together with the search space design. More concretely, we decouple the constrained optimization problem of Eq. (1) into three separate steps:

**Step 1.** Searching for an optimal search space under a specific constraint:

\[
\mathcal{A}^* = \min_{\mathcal{A} \subset \Omega} Q(\mathcal{A}; \mathcal{D}_{\text{val}}) \quad \text{s.t.} \quad |\mathcal{A}| \leq M, \tag{2}
\]

where \( Q \) is a measurement of search space to be elaborated in Sec. 2.3, and \( M \) represents the maximum search space size.

**Step 2.** Following one-shot NAS [12, 34], we encode the search space into a supernet and optimize its weights. This is formulated as solving the following problem:

\[
W_{\alpha^*}^* = \arg \min_W E_{\alpha \in \mathcal{A}^*} [\mathcal{L}(W_{\alpha}; \mathcal{D}_{\text{train}})], \tag{3}
\]

where \( W \) is the weights of the supernet, \( W_{\alpha} \) is the weights in \( W \) specified by the architecture \( \alpha \).

**Step 3.** After obtaining the well-trained supernet, we search for the optimal architecture via ranking the performance using the learned weights, which is formulated as:

\[
\alpha_{\mathcal{A}^*}^* = \arg \min_{\alpha \in \mathcal{A}^*} \mathcal{L}(W_{\alpha}^*; \mathcal{D}_{\text{val}}), \quad \text{s.t.} \quad g(\alpha) < c, \tag{4}
\]

where \( W_{\alpha}^* \) is the weights of architecture \( \alpha \) inherited from \( W_{\alpha^*}^* \).

### 2.2 Basic Search Space

We set up the general architectures of vision transformer by following the settings in ViT [10] and Swin-T [27] as shown in Fig. 1 (right). To be specific, given an input image, we uniformly split it into non-overlap patches. Those patches are linearly projected to vectors named patch embeddings. The embeddings are then fed into the transformer encoder, which performs most of the computation. At last, a fully connected layer is adopted for classification.

The transformer encoder consists of four sequential stages with progressively reduced input resolutions. Each stage contains blocks with the same embedding dimension. Therefore, stage \( i \) has two search dimensions: the number of blocks \( d_i \) and embedding dimension \( h_i \). Each block contains a window-based multi-head self-attention (WSA) module and a feed-forward network (FFN) module. We do not force the blocks in one stage to be identical. Instead, block \( j \) in the \( i \)th stage have several search dimensions including window size \( w_{ij} \), number of heads \( n_{ij} \), MLP ratio \( m_i \), \( Q-K-V \) embedding dimension \( q_{ij} \).

### 2.3 Searching the Search Space

**Space Quality Evaluation.** For a given space \( \mathcal{A} \), we propose to use \( E-T \) Error, denoted as \( Q(\mathcal{A}) \), for evaluating its quality. It is the average of two parts: expected error rate \( Q_e(\mathcal{A}) \) and top-tier error \( Q_t(\mathcal{A}) \). For a given space \( \mathcal{A} \), the definition of \( Q_e(\mathcal{A}) \) is given as:

\[
Q_e(\mathcal{A}) = \mathbb{E}_{\alpha \in \mathcal{A}, g(\alpha) < c}[e_{\alpha}], \tag{5}
\]

where \( e_{\alpha} \) is the top-1 error rate of architecture \( \alpha \) evaluated on ImageNet validation set, \( g(\cdot) \) is the computation cost function and \( c \) is the computation resource constraint. It measures the overall quality of a search space. In practice, we use the average error rate of \( N \) randomly sampled architectures to approximate the expectation term. The top-tier error \( Q_t(\mathcal{A}) \) is the average error rate of top 50 candidates under resource constraints, representing the performance upper bound of the search space.

**Once-for-all Supernet.** In constrast to RegNet [35] that trains hundreds of models with only a few epochs (~10) and uses their errors as a performance proxy of the fully trained networks, we encode the search space into a supernet and optimize it similar to AutoFormer [6]. Under such training protocol, we are able to train an once-for-all transformer supernet that serves as a reliable and efficient performance indicator.

**Searching the Search Space.** The search of search space mainly contains two iterative steps: 1) supernet optimization and 2) space evolution.
we remove the choices if they become less than zero. The detailed algorithm is also shown in Alg. 1.

$$\tau$$ where $$x$$

Algorithm 1 Searching the Search Space

**Input:** Infinite search space $$\Omega$$, max iteration $$T$$, threshold $$\tau$$ for subspace evolved, the subspaces $$S_1^{(0)}, \ldots, S_D^{(0)}$$, evolve steps for subspaces $$\gamma_1, \ldots, \gamma_D$$

**Output:** The most promising search space $$A^*$$

1. Initialize a search space $$A^{(0)}$$ from $$\Omega$$ with minimal prior human knowledge
2. while $$t \in [0, 1, \ldots, T]$$ do
   3. Optimize the weights $$W^{(t)}$$ of supernet corresponding to the space $$A^{(t)}$$
   4. Randomly sample $$N$$ architectures from the well converged supernet
   5. for space $$S_i^{(t)}$$ in $$S_1^{(t)}, \ldots, S_D^{(t)}$$ do
      6. Get the subspace $$\{\alpha | \alpha \in A^{(t)} \cap \alpha_i = v_i^{(t)}\}$$ for each choice $$v_i^{(t)} \in S_i^{(t)}$$, where $$\alpha_i$$ is the $$i$$th dimension value of the architecture $$\alpha$$
      7. Calculate the E-T Error of each subspace according to Eq. (5)
   8. Evolve the subspace and get $$S_i^{(t+1)}$$ according to Eq. (8)
   9. end for
10. $$A^{(t+1)} = S_1^{(t+1)} \times S_2^{(t+1)} \times \cdots \times S_D^{(t+1)}$$
11. end while

**Step 1.** For the search space $$A^{(t)}$$ of $$t$$th iteration, we encode it into a supernet. We adopt sandwich training [52] to optimize the weights of supernet, i.e., sampling the largest, the smallest, and two middle models for each iteration and fusing their gradients to update the supernet. Notably, we do not use inplace-distill since it leads to collapse for transformer training.

**Step 2.** We first decompose the search space $$A^{(t)}$$ by the search dimensions, defined in Sec. 2.2, and four stages. Denote their corresponding subspaces as $$S_1^{(t)}, S_2^{(t)}, \ldots, S_D^{(t)}$$, where $$D$$ is the product of the number of stage and the number of search dimensions. The search space $$A^{(t)}$$ now can be expressed as the Cartesian product of all search spaces:

$$A^{(t)} = S_1^{(t)} \times S_2^{(t)} \times \cdots \times S_D^{(t)}, \quad (6)$$

Then the search space $$A^{(t)}$$ could be evolved to a better one $$A^{(t+1)}$$, with the update of $$S_i^{(t)}$$. Particularly, for a certain subspace $$S_i^{(t)}$$, we first get the subspace $$\{\alpha | \alpha \in A^{(t)} \cap \alpha_i = v_i^{(t)}\}$$ for each choice $$v_i^{(t)} \in S_i^{(t)}$$, where $$\alpha_i$$ is the $$i$$th dimension value of the architecture $$\alpha$$. We then compute the E-T Errors of the subspaces. We hypothesize there is a strong correlation between the continuous choices and the E-T Error in most cases, so we fit a linear function to estimate its tendency for simplification:

$$y = wx + b, \quad (7)$$

where $$x \in \mathbb{R}$$ is the choices in $$S_i^{(t)}$$, and $$y \in \mathbb{R}$$ is the corresponding E-T Error. $$w$$ and $$b$$ are both scalar parameters. Then the new subspace $$S_i^{(t+1)}$$ is given as:

$$S_i^{(t+1)} = \{v_j^{(t)} - \frac{w}{\tau} \gamma_i \mid v_j^{(t)} \in S_i^{(t)}\}, \quad (8)$$

where $$\tau$$ is the threshold for search space evolving, $$\gamma_i$$ is the steps for search dimension $$S_i$$. Note that we remove the choices if they become less than zero. The detailed algorithm is also shown in Alg. 1.

### 2.4 Searching in the Searched Space

Once the space search is done, we perform neural architecture search over the searched space. The search pipeline contains two sequential steps: 1) supernet training without resource constraints, and 2) evolution search under resource constraints.

The **supernet training** follows the same train recipe in space search. Similar to AutoFormer [6] and BigNas [52], the maximal, minimal subnets and several random architectures are sampled randomly from the evolved search space in each iteration and their corresponding weights are updated. We provide more details in Sec. 4.1. The **evolution search** process follows the similar setting in SPOS [12]. Our objective in this paper is to maximize the classification accuracy while minimizing the model size and FLOPs. Notably, vision transformer does not include any Batch Normalization [19],...
3 Analysis and Discussion

In this section we provide analysis and discussion according to the space searching process. We hope that our observations can shed lights on both manual design of vision transformer architecture and the search space design for NAS methods. To minimize the prior knowledge used in search space design, we initialize the search space by setting a same search space for different stages. We provide the detailed search space and its evolution process in Fig. 2. Fig. 3 shows a direct illustration of space quality of the searched space focusing on the number of blocks.

The third stage is most important and increasing blocks in the third stage leads to better performance. As shown in the first row of Fig. 2, the blocks of stage 3 is constantly increasing from step 0 to step 2. We also observe that there are more blocks in stage 3 in some famous CNN networks, such as ResNet [14], EfficientNet [39], which share the same architecture characteristics with our finding in transformers. As a result, it seems that the third stage is the most important one. On the other hand, contrary to stage 3, the number of blocks decrease in stage 1,2,4 showing that these stages should include fewer layers.

Shallow layers should use a small window size while deep layers should use a larger one. As shown in the last row of Fig. 2, with the search steps increasing, the window size space decreases in stage 1 while increases in stage 3. It shows that shallow layers tend to have smaller window size while deep layers tend to have larger one. We assume that transformers increase focusing region with deeper layers. To verify this, we visualize the attention maps (top row in Fig. 4 (b,c) ) in different layers and...
Figure 4: Illustration of attention maps of some selected layers ((b)-(e)), and the average relative distance of a query and its top-k keys with highest attention ((f), (g)). For attention maps (“3-7" represents the “stage 3 - block 7”, while (c)-(e) follow the rule), the reference position is on left top, marked with +. The numbers 1-5 on attention map subpatches represent the top-5 highest attention. Besides, bottom row shows top-5/10 average relative distances of some selected layers. The orange lines | represent the median and the green triangles ▲ represent the mean.

draw a box plot (bottom row in Fig. 4 (f,g)) to show the relative distances from the target position to its top-5/10 focusing positions. The attention maps show that the highest attentions (noted by numbers) expand with deeper layer, which is illustrated from (b) to (c). As for the average relative distance, deeper layers have larger medians and means, indicating larger focusing regions. The above observations are consistent with our assumption and provide solid evidence for it.

**Deeper layer should use larger MLP ratio.** Traditionally, all layers use the same MLP ratio. However, the third row of Fig. 2 illustrates that shallow layers should use small MLP ratio to result in better performance while deeper ones should use larger MLP ratio.

**Q-K-V dimension should be smaller than the embedding dimension.** In the original transformer block [41, 10], the Q-K-V dimension is the same as the embedding dimension. However, comparing the second row with the third row of Fig. 2, we find that the searched space has a slightly smaller Q-K-V dimension than embedding dimension, shown from stage 1 to stage 4. The gap between them are more obvious in the deeper layers. We conjecture the reason is that there are many heads with similar features in deeper layers. Therefore, a relative small Q-K-V dimension is able to have great visual representation ability. We present more details in the supplementary materials.

4 Experiments

4.1 Implementation Details

**Space searching.** The initial search space is presented in Fig. 2, the number of blocks and embedding dimension are set to {2, 3, 4} and {224, 256, 288} respectively for all stages, while the window size, number of heads, MLP ratio, Q-K-V dimensions are set to {7, 14}, {3, 3.5, 4}, {7, 8, 9}, {224, 256, 288} respectively for all blocks. We set the total iteration of space searching to 3, while setting the steps in Eq. (8) for block number, embed dim, MLP ratio, Q-K-V dim, windows size to 1, 64, 0.5, 64, 7, respectively. We set the threshold $\tau$ of space evolution in Eq. (8) as 0.4.

**Supernet training.** Similar to DeiT [40], we train the supernet with the following settings: AdamW [28] optimizer with weight decay 0.05, initial learning rate $1 \times 10^{-3}$ and minimal learning rate $1 \times 10^{-5}$ with cosine scheduler, 20 epochs warmup, batch size of 1024, 0.1 label smoothing, and stochastic depth with drop rate 0.2. We also adopt sandwich strategy as described in Sec. 2.3. The models are trained for 300 epochs with 16 Nvidia Tesla 32G V100 GPUs.
Table 1: S3 performance on ImageNet in comparison with state-of-the-arts. We group the models according to their parameter sizes.

<table>
<thead>
<tr>
<th>Models</th>
<th>Top-1 Acc.</th>
<th>Top-5 Acc.</th>
<th>#Params.</th>
<th>FLOPs</th>
<th>Res.</th>
<th>Model Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50 [14]</td>
<td>76.2%</td>
<td>92.9%</td>
<td>25.5M</td>
<td>4.1G</td>
<td>224²</td>
<td>CNN</td>
</tr>
<tr>
<td>RegNetY-4GF [35]</td>
<td>80.0%</td>
<td>-</td>
<td>21.4M</td>
<td>4.0G</td>
<td>224²</td>
<td>CNN</td>
</tr>
<tr>
<td>EfficientNet-B4 [39]</td>
<td>82.9%</td>
<td>95.7%</td>
<td>19.3M</td>
<td>4.2G</td>
<td>380²</td>
<td>CNN</td>
</tr>
<tr>
<td>T2T-ViT-14 [53]</td>
<td>80.7%</td>
<td>-</td>
<td>21.5M</td>
<td>6.1G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>DeiT-S [40]</td>
<td>79.9%</td>
<td>95.0%</td>
<td>22.1M</td>
<td>4.7G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>ViT-S/16 [10]</td>
<td>78.8%</td>
<td>-</td>
<td>22.1M</td>
<td>4.7G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>Swin-T [27]</td>
<td>81.3%</td>
<td>-</td>
<td>28.3M</td>
<td>4.5G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>S3-T (Ours)</td>
<td>82.1%</td>
<td>95.8%</td>
<td>28.1M</td>
<td>4.7G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>ResNet152 [14]</td>
<td></td>
<td></td>
<td>60M</td>
<td>11G</td>
<td>224²</td>
<td>CNN</td>
</tr>
<tr>
<td>EfficientNet-B7 [39]</td>
<td>84.3%</td>
<td>97.0%</td>
<td>66M</td>
<td>37G</td>
<td>600²</td>
<td>CNN</td>
</tr>
<tr>
<td>BoTNet-S1-110 [38]</td>
<td>82.8%</td>
<td>96.4%</td>
<td>55M</td>
<td>10.9G</td>
<td>224²</td>
<td>CNN + Trans</td>
</tr>
<tr>
<td>T2T-ViT-24 [53]</td>
<td>82.2%</td>
<td>-</td>
<td>64M</td>
<td>15G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>Swin-S [27]</td>
<td>83.0%</td>
<td>-</td>
<td>50M</td>
<td>8.7G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>S3-S (Ours)</td>
<td>83.7%</td>
<td>96.4%</td>
<td>50M</td>
<td>9.5G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>S3-S-384 (Ours)</td>
<td>84.5%</td>
<td>96.8%</td>
<td>50M</td>
<td>33G</td>
<td>384²</td>
<td>Transformer</td>
</tr>
<tr>
<td>RegNetY-16GF [35]</td>
<td>80.4%</td>
<td>-</td>
<td>83M</td>
<td>16G</td>
<td>224²</td>
<td>CNN</td>
</tr>
<tr>
<td>ViT-B/16 [10]</td>
<td>79.7%</td>
<td>-</td>
<td>86M</td>
<td>18G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>DeiT-B [40]</td>
<td>81.8%</td>
<td>95.6%</td>
<td>86M</td>
<td>18G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>Swin-B [27]</td>
<td>83.3%</td>
<td>-</td>
<td>88M</td>
<td>15.4G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>Swin-B-384 [27]</td>
<td>84.2%</td>
<td>-</td>
<td>88M</td>
<td>47G</td>
<td>384²</td>
<td>Transformer</td>
</tr>
<tr>
<td>S3-B (Ours)</td>
<td>84.0%</td>
<td>96.6%</td>
<td>71M</td>
<td>13.6G</td>
<td>224²</td>
<td>Transformer</td>
</tr>
<tr>
<td>S3-B-384 (Ours)</td>
<td>84.7%</td>
<td>96.9%</td>
<td>71M</td>
<td>46G</td>
<td>384²</td>
<td>Transformer</td>
</tr>
</tbody>
</table>

4.2 Ablation Study

Effectiveness of once-for-all supernet sampling. To verify the effectiveness, we randomly sample 10 architectures from the well-trained supernet with the inherited weights, then compare the ImageNet classification performance of (1) models with only inherited weights, (2) models with finetuning, and (3) models retrained from scratch. As shown in Fig. 5, the architecture with inherited weights has comparable or even better performance when compared with the same architecture with retraining or finetuning. This phenomenon shows that our sampled architectures provide a precise and reliable performance proxy for space quality measurement.

Effectiveness of search space evolution. We conduct two experiments. In the first experiment, we randomly sample 1, 000 architectures with inherited weights from a search space, then compute and plot error empirical distribution, as in [35], of them. We denote the initial space, the one-step evolved space and the two-step evolved space by Space A, Space B and Space C respectively. As shown in Fig. 6, the space C has an obviously better overall quality compared to A and B. In the second experiment, we use evolution algorithm to search for top architectures in each space. Tab. 5 illustrates that the space C has a much better top architectures, being 16.6%/4.1% better than that in the spaces A and B. These results verify the effectiveness of search spaces evolution.
We compare our S3 with state-of-the-art CNN models, vision transformers and hybrid CNN-transformer models. The results are reported in Tab. 1. Our proposed models S3-tiny/small/base (T/S/B for short) achieve 82.1%/83.7%/84.0% top-1 accuracy on ImageNet [9] using 224 input images. Compared with vision transformer and hybrid models, our method obtains superior performance using similar FLOPs and parameters. For example, when considering parameters as the main constraint, we find a architecture that has 0.6% higher top-1 accuracy than Swin Transformer [27] while using 42% fewer parameters as shown in Fig. 7. Compared to CNN models, our S3 models are also very competitive, especially for small parameter size. The proposed S3-S-384 (84.5%) performs slightly better than EfficientNet-B7 (84.3%) [39] while using 24% fewer parameters and 11% fewer FLOPs. For base models with fewer computational cost, S3-B also achieves superior performance to state-of-the-art methods, such as RegNet [35] and ViT [10].

### 4.3 Image Classification

We compare our searched models with state-of-the-arts on ImageNet [9]. As described in Sec. 2.1 and Sec. 4.1, we first evolve the search space, then optimize the weights of supernet and finally search on the supernet. For the experiments of high resolution images, the models are finetuned with 384 inputs for 10 epochs, using an AdamW [28] optimizer with a constant learning rate of $10^{-5}$ and weight decay of $10^{-8}$.

We compare our S3 with state-of-the-art CNN models, vision transformers and hybrid CNN-transformer models. The results are reported in Tab. 1. Our proposed models S3-tiny/small/base (T/S/B for short) achieve 82.1%/83.7%/84.0% top-1 accuracy on ImageNet [9] using 224 input images. Compared with vision transformer and hybrid models, our method obtains superior performance using similar FLOPs and parameters. For example, when considering parameters as the main constraint, we find a architecture that has 0.6% higher top-1 accuracy than Swin Transformer [27] while using 42% fewer parameters as shown in Fig. 7. Compared to CNN models, our S3 models are also very competitive, especially for small parameter size. The proposed S3-S-384 (84.5%) performs slightly better than EfficientNet-B7 (84.3%) [39] while using 24% fewer parameters and 11% fewer FLOPs. For base models with fewer computational cost, S3-B also achieves superior performance to state-of-the-art methods, such as RegNet [35] and ViT [10].

### 4.4 Vision and Vision-Language Downstream Tasks

To further evaluate the generalizability of the discovered architectures, we transfer them to vision and vision-language downstream tasks. We conduct experiments on COCO [24] for object detection, ADE20K [36] for semantic segmentation and VQA 2.0 [34] for visual question answering, respectively. The training and inference settings for object detection and semantic segmentation are the same as those in Swin transformer [27], while for visual language answering task, the settings are aligned with SOHO [18]. More experimental details are elaborated in supplementary materials.

**Object Detection.** As shown in Tab. 2, S3-T outperforms ResNet-50 [14] by +4.2 box AP and +3.7 mask AP under 3× learning schedule. Compared to DeiT-S [40], S3-T achieves +2.5 higher box AP and +2.4 higher mask AP. Increasing the model size and FLOPs, S3-S and S3-B surpass ResNeXt-101 [48] by +4.5 box AP, +3.9 mask AP and +4.2 box AP, +3.7 mask AP respectively. Compared to Swin-S and Swin-B, S3-S and S3-B outperform them by +0.8 box AP, +0.8 mask AP and +0.6 box AP, +0.4 mask AP respectively. All comparisons are conducted under the similar model constraints.

### Table 2: Results on COCO object detection using Cascaded Mask R-CNN. We provide two settings with FPN (1×: 12 epochs; 3×: 36 epochs). Results are taken from [44] and its official repository.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>#Params (M)</th>
<th>#FLOPs (G)</th>
<th>AP50</th>
<th>AP60</th>
<th>AP75</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50 [14]</td>
<td>82</td>
<td>739</td>
<td>41.2</td>
<td>59.4</td>
<td>45.0</td>
</tr>
<tr>
<td>DeiT-S [44]</td>
<td>80</td>
<td>889</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Swin-T [27]</td>
<td>86</td>
<td>745</td>
<td>48.1</td>
<td>67.1</td>
<td>52.2</td>
</tr>
<tr>
<td>S3-T (Ours)</td>
<td>86</td>
<td>748</td>
<td>48.4</td>
<td>67.8</td>
<td>52.3</td>
</tr>
</tbody>
</table>

Figure 7: Comparison with state-of-the-art models under different constraints.
size and FLOPs. Under the $1 \times$ training schedule, our S3 model is also superior to other backbones, demonstrating its robustness and good transferability.

**Semantic Segmentation.** We report the mIoU score on ADE20K [56] validation set in Tab. 3. It can be seen that our S3 models consistently achieve better performance than ResNet [14], DeiT [40] and Swin [27]. Specifically, S3-T is $+3.5$ mIoU superior to ResNet-50. Compared to other transformer-based models, S3-T is $+2.42$ mIoU higher than DeiT-S, and $+0.46$ mIoU higher than Swin-T. S3-S outperforms ResNet-101 with a large margin ($+4.46$). It also surpasses DeiT-B and Swin-S by $+3.63$ and $+0.44$ mIoU, respectively. Note that we tried to add deconvolution layers to build a hierarchical structure for DeiT, but did not get performance improvements.

**Visual Question Answering.** As shown in Tab. 4, we report the accuracy on the test-dev and test-std set of VQA v2.0 dataset [54] for different vision backbones. Our model S3-T brings more than $+2\%$ accuracy gains over ResNet-101 [14], and surpassing Swin-T [27] by around $0.6\%$ accuracy under aligned model complexity. All the above experiments demonstrate the good generality of the searched models as well as proving the effectiveness of the searched design space of vision transformer.

### 5 Related Work

**Vision Transformer.** Originally designed for natural language processing, Transformer has been recently applied to vision tasks [4, 55, 5, 46]. ViT [10] is the first one that introduces a pure transformer architecture for visual recognition. Alleviating the data hungry and computational issues, DeiT [40] introduces a series of training strategies to remedy the limitations. Some further works resort to feature pyramids to purify multi-scale features and model global information [44, 15, 31]. Swin Transformer [27] introduces non-overlapping window partitions and restricts the calculation of self-attention within each window, leading to a linear runtime complexity w.r.t. token numbers.

**Search Space.** There is a wide consensus that a good search space is crucial for NAS. Regarding CNN, the current search space, such as cell-based space [26, 7, 49] and MBConv-based space [3, 32, 51], is pre-defined and fixed during search. Several recent works [17, 29, 30, 33, 25, 23] perform search space shrinking to find a better compact one. Moreover, NSENet [8] proposes an evolving search space algorithm for MBConv space optimization. RegNet [35] presents a new paradigm for understanding design space and discovering design principles. However, the exploration of vision transformer space is still limited, and our work is the first along the direction.

**Search Algorithm.** There has been an increasing interest in NAS for automating network design [11, 20]. Early approaches use either reinforcement learning [57, 58] or evolution algorithms [47, 36]. These approaches require training thousands of architecture candidates from scratch. Most recent works resort to the one-shot weight sharing strategy to amortize the searching cost [22, 34, 1, 12]. The key idea is to train an over-parameterized supernet where all subnets share weights. For transformer, there are few works employing NAS to improve architectures. Evolved Transformer [37] searches the inner architecture of transformer block. HAT [43] proposes to design hardware-aware transformers with NAS to enable low-latency inference on resource-constrained hardware platforms. BossNAS [21] explores hybrid CNN-Transformers with block-wisely self-supervised. The most related work is AutoFormer [6] which provides a simple yet effective framework of vision transformer search. It could achieve once-for-all training with newly proposed weight entanglement. In this work, we focus on both the search space design and architectures of vision transformer.

### Table 3: Results on ADE20K Semantic Segmentation using UperNet. $\dagger$: reproduced by the official code. $\ast$: without adding deconvolution in heads.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>#Param (M)</th>
<th>FLOPs (G)</th>
<th>mIoU (%)</th>
<th>mIoU (ms+flip %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UperNet</td>
<td>ResNet-50 [14]</td>
<td>67</td>
<td>931</td>
<td>42.05</td>
<td>42.78</td>
</tr>
<tr>
<td>UperNet</td>
<td>DeiT-S$^*$ [40]</td>
<td>52</td>
<td>1099</td>
<td>43.15</td>
<td>43.85</td>
</tr>
<tr>
<td>UperNet</td>
<td>Swin-T [27]</td>
<td>60</td>
<td>945</td>
<td>44.51</td>
<td>45.81</td>
</tr>
<tr>
<td>UperNet</td>
<td>S3-T (Ours)</td>
<td>60</td>
<td>954</td>
<td><strong>44.87</strong></td>
<td><strong>46.27</strong></td>
</tr>
<tr>
<td>UperNet</td>
<td>ResNet-101 [14]</td>
<td>86</td>
<td>1029</td>
<td>43.82</td>
<td>44.85</td>
</tr>
<tr>
<td>UperNet</td>
<td>DeiT-B$^+$ [40]</td>
<td>121</td>
<td>2772</td>
<td>44.09</td>
<td>45.68</td>
</tr>
<tr>
<td>UperNet</td>
<td>Swin-S$^*$ [27]</td>
<td>81</td>
<td>1038</td>
<td>47.60</td>
<td>48.87</td>
</tr>
<tr>
<td>UperNet</td>
<td>S3-S (Ours)</td>
<td>81</td>
<td>1071</td>
<td><strong>48.04</strong></td>
<td><strong>49.31</strong></td>
</tr>
</tbody>
</table>

### Table 4: Results on VQA v2.0 dataset.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>#Param (M)</th>
<th>FLOPs (G)</th>
<th>mIoU (%)</th>
<th>mIoU (ms+flip %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50 [14]</td>
<td>4.1</td>
<td>26</td>
<td>64.65</td>
<td>65.12</td>
</tr>
<tr>
<td>ResNet-101 [14]</td>
<td>7.8</td>
<td>45</td>
<td>65.44</td>
<td>65.71</td>
</tr>
<tr>
<td>Swin-T [27]</td>
<td>4.5</td>
<td>29</td>
<td>67.13</td>
<td>67.40</td>
</tr>
<tr>
<td>S3-T (Ours)</td>
<td>4.6</td>
<td>29</td>
<td><strong>67.84</strong></td>
<td><strong>68.00</strong></td>
</tr>
</tbody>
</table>

### Table 5: Comparisons of searched architectures in different spaces.

<table>
<thead>
<tr>
<th>Space</th>
<th>#Param (M)</th>
<th>FLOPs (G)</th>
<th>Top-1 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>28M</td>
<td>4.7</td>
<td>66.5</td>
</tr>
<tr>
<td>B</td>
<td>29M</td>
<td>4.7</td>
<td>78.0</td>
</tr>
<tr>
<td>C</td>
<td>28M</td>
<td>4.7</td>
<td>82.1</td>
</tr>
</tbody>
</table>
6 Conclusion

In this work, we propose to search the search space of vision transformer. The central idea is to gradually evolve different search dimensions guided by their E-T Error computed using a weight-sharing supernet. We also provide analysis on vision transformer, which might be helpful for understanding and designing transformer architectures. The search models, S3, achieve superior performance to recent prevalent ViT and Swin transformer model families under aligned settings. We further show their robustness and generality on downstream tasks. In further work, we hope to investigate the usage of S3 in CNN search space design.

Acknowledgements and Disclosure of Funding

Thank Hongwei Xue for providing helps and discussions on vision-language tasks. No external funding was received for this work.

References

[10] 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. ICLR, 2021. 1, 2, 3, 6, 7, 8, 9


[23] Xiang Li, Chen Lin, Chuming Li, Ming Sun, Wei Wu, Junjie Yan, and Wanli Ouyang. Improving one-shot nas by suppressing the posterior fading. In *CVPR*, 2020.


[34] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *ICML*, 2018.


[56] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In CVPR, 2017. 8, 9

[57] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. ICLR, 2016. 9

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In the supplementary materials
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In the supplementary materials
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [No]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]