Searching the Search Space of Vision Transformer

Anonymous Author(s)
Affiliation
Address
email

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 are open-sourced (anonymously) 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 [40], modern state-of-the-art models, such as ViT [9] and DeiT [39], are able to achieve competitive performance on image classification and downstream vision tasks [12, 19] compared to *CNN* models. There are fast-growing efforts on manually designing more effective architectures to further unveil the power of vision transformer [26, 51, 14].

Neural architecture search (NAS) as a powerful technique for automating the design of networks has shown its superiority to manual design [56, 38]. 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, 15, 48]. Researchers have devoted ample efforts to the design of *CNN* space [44, 11, 41]. 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 [34] focusing on the overall quality, while the "T" part considers the quality of top-tier architectures. Different from [34] that trains each subnet for only a few epochs, we train a once-for-all supernet to obtain a large amount of well-trained subnets, providing a more reliable proxy for performance estimation.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

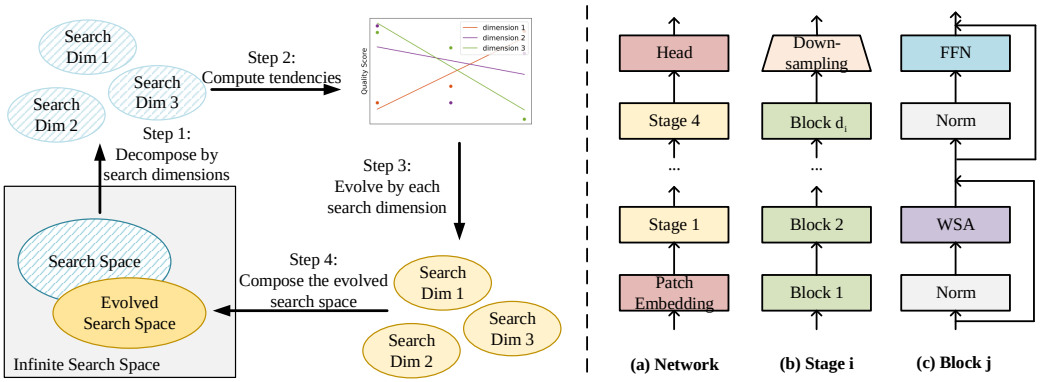

Figure 1: **Left:** The pipeline of searching the search space. **Right:** The general vision transformer architecture explored in this paper.

For the second question, as shown in Fig. 1 we decouple 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 prefer a small window size while deep layers prefer 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 could 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 [9] and Swin [26] 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:

$$\alpha^*_{\mathcal{A}} = \arg\min_{\alpha \in \mathcal{A}} \mathcal{L}(W^*_\alpha; \mathcal{D}_{\text{val}}) \quad s.t. \quad W^*_\alpha = \arg\min_{W_\alpha} \mathcal{L}(W_\alpha; \mathcal{D}_{\text{train}}), \quad g(\alpha) < c, \tag{1}$$

where W_α is the weights of the network architecture α, $\mathcal{A}$ is a predefined search space, $\mathcal{L}(\cdot)$ is the loss function, $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$ 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 $\mathcal{A}$ is an infinite space Ω that contains all possible architectures. In practice, $\mathcal{A}$ is usually a small subspace of Ω 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:

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

where $\mathcal{Q}$ 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 [11, 33], we encode the search space into a supernet and optimize its weights. This is framed as solving the following problem:

$$W^*_{\mathcal{A}^*} = \arg\min_{W} \mathbb{E}_{\alpha \in \mathcal{A}^*}[\mathcal{L}(W_\alpha; \mathcal{D}_{\text{train}})], \tag{3}$$

where W is the weights of the supernet, W_α is the weights in W specified by the architecture α.

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:

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

where W^*_α is the weights of architecture α inherited from $W^*_{\mathcal{A}^*}$.

2.2 Basic Search Space

We set up the general architectures of vision transformer by following the settings in ViT [9] and Swin-T [26] 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 feedforward network (FFN) module. We do not force the number of blocks in different stages to be identical. Instead, block j in the ith stage have several search dimensions including window size w^i_j, number of heads n^i_j, MLP ratio m_i, Q-K-V embedding dimension q^i_j.

2.3 Searching the Search Space

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

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

where e_α is the top-1 error rate of architecture α 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 $\mathcal{Q}_t(\mathcal{A})$ 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. Different from RegNet [34] that trains hundreds of models with only a few epochs ($\sim$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 BigNAS [50] and HAT [42]. 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.

Step 1. For the search space $\mathcal{A}^{(t)}$ of tth iteration, we encode it into a supernet. The weight sharing strategy of the supernet is similar to [42]. We adopt sandwich training [50] 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 $\mathcal{A}^{(t)}$ by the search dimensions, defined in Sec .2.2, and four stages. Denote their corresponding subspaces as $S^{(t)}_1, S^{(t)}_2, \cdots, S^{(t)}_D$, where D is the product

Algorithm 1 Searching the Search Space

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

Output: The most promising search space $\mathcal{A}^*$

Initialize a search space $\mathcal{A}^{(0)}$ from Ω with minimal prior human knowledge
while $t \in [0, 1, \cdots, T]$ **do**
    Optimize the weights $W_{\mathcal{A}}^{(t)}$ of supernet corresponding to the space $\mathcal{A}^{(t)}$
    Randomly sample N architectures from the well converged supernet
    for space $S_i^{(t)}$ in $S_1^{(t)}, \cdots, S_D^{(t)}$ **do**
        Get the subspace $\{\alpha | \alpha \in \mathcal{A}^{(t)} \cap \alpha_i = v_l^{(t)}\}$ for each choice $v_l^{(t)} \in S_i^{(t)}$, where α_i is the ith dimension value of the architecture α
        Calculate the *E-T Error* of each subspace according to Eq. (5)
        Evolve the subspace and get $S_{D_i}^{(t+1)}$ according to Eq. (8)
    end for
    $\mathcal{A}^* = \mathcal{A}^{(t+1)} = S_1^{(t+1)} \times S_2^{(t+1)} \times \cdots \times S_D^{(t+1)}$
end while

of the number of stage and the number of search dimensions. The search space $\mathcal{A}^{(t)}$ now can be expressed as the Cartesian product of all search spaces:

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

Then the search space $\mathcal{A}^{(t)}$ could be evolved to a better one $\mathcal{A}^{(t+1)}$, with the update of $S_i^{(t)}$. Particularly, For a certain subspace $S_i^{(t)}$, we first get the subspace $\{\alpha | \alpha \in \mathcal{A}^{(t)} \cap \alpha_i = v_l^{(t)}\}$ for each choice $v_l^{(t)} \in S_i^{(t)}$, where α_i is the ith dimension value of the architecture α. Then we 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:

$$\mathbf{y} = w\mathbf{x} + b, \tag{7}$$

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

$$S_i^{(t+1)} = \left\{ v_j^{(t)} - \left\lfloor \frac{w}{\tau} \right\rfloor \gamma_i \;\middle|\; v_j^{(t)} \in S_i^{(t)} \right\}, \tag{8}$$

where τ is the threshold for search space evolving, γ_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 SPOS [11] and BigNas [50], 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 [11]. 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 [18], hence the evolution process is much faster than CNN. We provide more details in the supplementary materials.

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

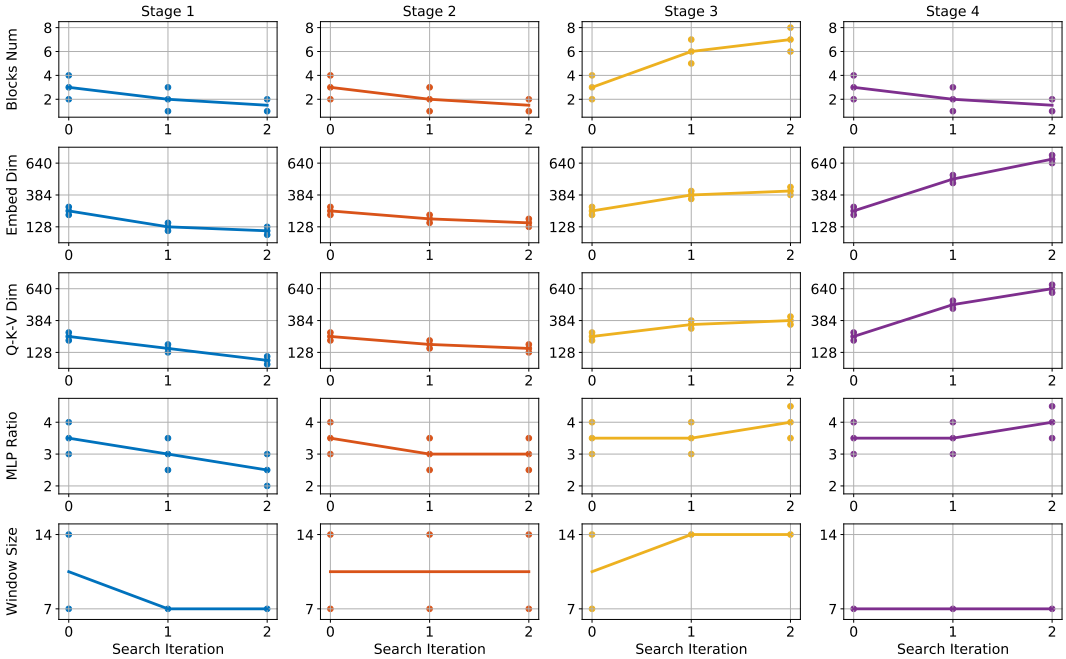

Figure 2: Visualization of space search process. Columns and rows represent different stages and search dimensions. The dots in each sub-figure means the choices of that search dimension. We plot addition lines for better viewing the change tendency of dimensions.

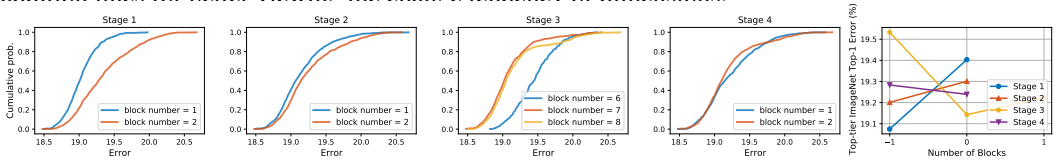

Figure 3: The left 4 subplots present the empirical distribution function of different stages of the searched space focusing on the number of blocks. The rightest is the top-tier error of different stages.

search space design for NAS methods. To minimize the prior knowledge used in search space design, we initialize the search space by setting the same search space for different stages. We provide the detailed search space and its evolution process in Fig. 2. Fig. 3 shows an direct illustration of space quality of the searched space focusing on the number of blocks.

The third stage is most important and increasing blocks of it contribute 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 [13], EfficientNet [38], which share the same architecture characteristics with our finding in tranformers. As a result, it seems that the third stage is the most important one. On the other hand, contrary to stage 3, block decreasing in stage 1,2,4 with the search steps shows that these stages should include less layers, while increasing the layers of stage 3 to get better performance.

Shallow layers prefer a small window size while deep layers prefer a large one. As shown in the last row of Fig. 2, with search steps increasing, the window size decreases in stage 1 while increases in stage 3. It shows that shallow layers tend to have smaller window size while deep layers attend 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 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 illustrated from (b) to (c). As for the average relative distance, deeper layers have larger medians and means, indicating large focusing regions. Both of them are consistent with our assumption, and verify that the deeper layers prefer larger window size.

Deeper layer, 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 deepers use larger one. It gives our insight to design different MLP ratio.

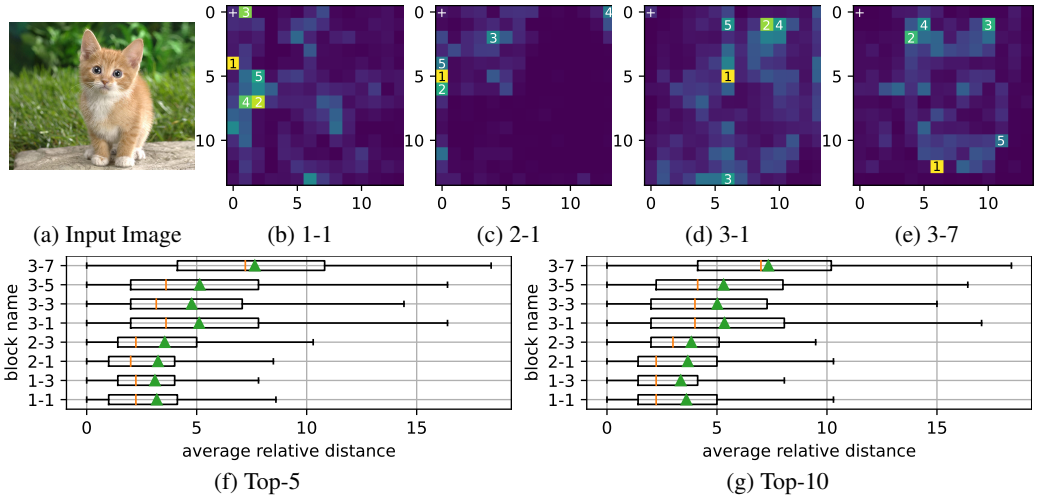

Figure 4: Illustration of attention maps of some selected layers ((b)-(e)), and the average relative distance of a query and its top-k keys with highest attention ((f), (g)). For attention maps ("1-1" represents the "block 1 - stage 1", while (c)-(e) follow the rule), the reference position is on left 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.

Q-K-V dimension could smaller than the embedding dimension. In the original transformer block [40, 9], 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. The reason might be that there are many similar feature maps in each head in deeper layers, which induces us to use less heads, further to use smaller Q-K-V dimension than embedding dimension. 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 τ of space evolution in Eq. (8) as 0.4.

Supernet training. Similar to DeiT [39], we train the supernet with the following settings: AdamW [27] optimizer with weight decay 0.05, initial learning rate 1×10^{-3} and minimal learning rate 1×10^{-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 describe in Sec. 2.3. The models are trained for 300 epochs with 16 Nvidia Tesla 32G V100 GPUs.

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.

Table 1: S3 performance on ImageNet in comparison with state-of-the-arts. We group the models according to their parameter sizes.

Models	Top-1 Acc.	Top-5 Acc.	#Params.	FLOPs	Res.	Model Type
ResNet50 [13]	76.2%	92.9%	25.5M	4.1G	224^2	CNN
RegNetY-4GF [34]	80.0%	-	21.4M	4.0G	224^2	CNN
EfficietNet-B4 [38]	82.9%	95.7%	19.3M	4.2G	380^2	CNN
T2T-ViT-14 [51]	80.7%	-	21.5M	6.1G	224^2	Transformer
DeiT-S [39]	79.9%	95.0%	22.1M	4.7G	224^2	Transformer
ViT-S/16 [9]	78.8%	-	22.1M	4.7G	224^2	Transformer
Swin-T [26]	81.3%	-	28.3M	4.5G	224^2	Transformer
S3-T (Ours)	**82.1%**	**95.8%**	**28.1M**	**4.7G**	224^2	Transformer
ResNet152 [13]	78.3%	94.1%	60M	11G	224^2	CNN
EfficietNet-B7 [38]	84.3%	97.0%	66M	37G	600^2	CNN
BoTNet-S1-110 [37]	82.8%	96.4%	55M	10.9G	224^2	CNN + Trans
T2T-ViT-24 [51]	82.2%	-	64M	15G	224^2	Transformer
Swin-S [26]	83.0%	-	50M	8.7G	224^2	Transformer
S3-S (Ours)	**83.7%**	**96.4%**	**50M**	**9.5G**	224^2	Transformer
S3-S-384 (Ours)	**84.5%**	**96.8%**	**50M**	**33G**	384^2	Transformer
RegNetY-16GF [34]	80.4%	-	83M	16G	224^2	CNN
ViT-B/16 [9]	79.7%	-	86M	18G	224^2	Transformer
DeiT-B [39]	81.8%	95.6%	86M	18G	224^2	Transformer
Swin-B [26]	83.3%	-	88M	15.4G	224^2	Transformer
Swin-B-384 [26]	84.2%	-	88M	47G	384^2	Transformer
S3-B (Ours)	**84.0%**	**96.6%**	**71M**	**13.6G**	224^2	Transformer
S3-B-384 (Ours)	**84.7%**	**96.9%**	**71M**	**46G**	384^2	Transformer

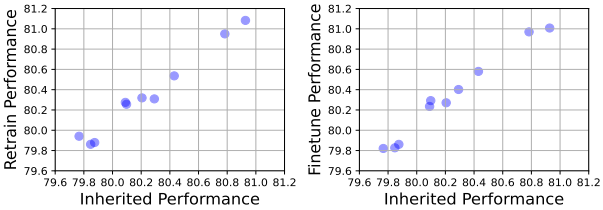

Figure 5: Comparison of subnets with inherited weights, fine-tuned and trained from-scratch.

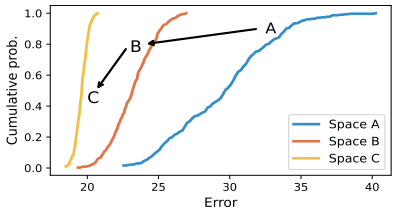

Figure 6: Error empirical distribution function of different search spaces.

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 [34], 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.

4.3 Image Classification

We compare our searched models with state-of-the-arts on ImageNet [8]. 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. Notably, the searched models inherit weights from the supernet directly, without extra retraining and post-processing. For the experiments of high resolution images, the models are finetuned with 384^2 inputs for 10 epochs, using an AdamW [27] 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 [8] using 224^2 input

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 [43] and its official repository.

Backbone	#Params (M)	#FLOPs (G)	Cas. Mask R-CNN w/ FPN 1×						Cas. Mask R-CNN w/ FPN 3×					
			AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
ResNet-50 [13]	82	739	41.2	59.4	45.0	35.9	56.6	38.4	46.3	64.3	50.5	40.1	61.7	43.4
DeiT-S [43]	80	889	-	-	-	-	-	-	48.0	67.2	51.7	41.4	64.2	44.3
Swin-T [26]	86	745	48.1	67.1	52.2	41.7	64.4	45.0	50.4	69.2	54.7	43.7	66.6	47.3
S3-T(Ours)	86	748	**48.4**	**67.8**	**52.3**	**42.0**	**64.8**	**45.2**	**50.5**	**69.4**	**54.9**	**43.8**	**66.8**	**47.5**
ResNeXt-101-32 [46]	101	819	44.3	62.7	48.4	38.3	59.7	41.2	48.1	66.5	52.4	41.6	63.9	45.2
Swin-S [26]	107	838	50.3	69.6	54.8	43.4	66.7	47.0	51.8	70.4	56.3	44.7	67.9	48.5
S3-S(Ours)	107	855	**50.8**	**70.1**	**55.0**	**43.9**	**67.5**	**47.2**	**52.6**	**71.5**	**56.9**	**45.5**	**68.8**	**49.4**
ResNeXt-101-64 [46]	140	972	45.3	63.9	49.6	39.2	61.1	42.2	48.3	66.4	52.3	41.7	64.0	45.1
Swin-B [26]	145	982	50.5	69.5	55.0	43.5	66.9	46.9	51.9	70.7	56.3	45.0	68.2	48.8
S3-B(Ours)	128	947	**50.8**	**69.7**	**55.2**	**43.7**	**67.1**	**47.0**	**52.5**	**71.2**	**57.1**	**45.4**	**68.8**	**49.1**

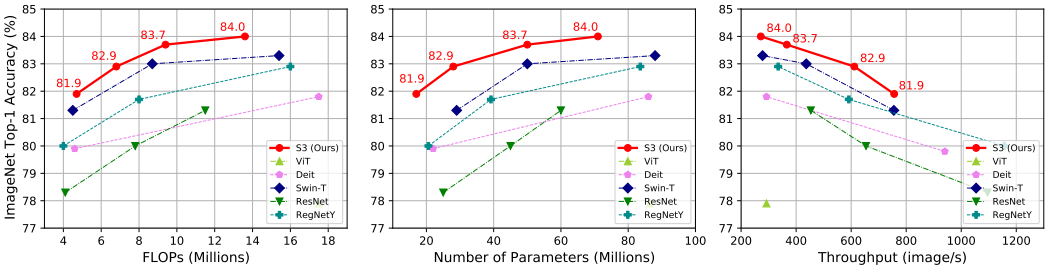

Figure 7: Comparison with state-of-the-art models under different constraints.

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 could get model that has 0.6% higher top-1 accuracy than Swin Transformer [26] 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%) [38] 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 [34] and ViT [9].

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 [23] for object detection, ADE20K [54] for semantic segmentation and VQA 2.0 [52] for visual question answering, respectively. The training and inference settings for object detection and semantic segmentation are the same as those in Swin transformer [26], while for visual language answering task, the settings are aligned with SOHO [17]. More experimental details are elaborated in supplementary materials.

Object Detection. As shown in Tab. 2, S3-T outperforms ResNet-50 [13] by +4.2 box AP and +3.7 mask AP under 3× learning schedule. Compared to DeiT-S [39], 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 [46] 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 size and FLOPs. Under the 1× 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 [54] validation set in Tab. 3. It can be seen that our S3 models consistently achieve better performance than ResNet [13], DeiT [39] and Swin [26]. 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.

Table 3: Results on ADE20K Semantic Segmentation using UperNet. †: reproduced by the official code. ∗: without adding deconvolution in heads.

Method	Backbone	#param (M)	FLOPs (G)	mIoU (%)	mIoU (ms+flip %)
UperNet	ResNet-50 [13]	67	951	42.05	42.78
UperNet	DeiT-S* [39]	52	1099	43.15	43.85
UperNet	Swin-T [26]	60	945	44.51	45.81
UperNet	**S3-T(Ours)**	60	954	**44.87**	**46.27**
UperNet	ResNet-101 [13]	86	1029	43.82	44.85
UperNet	DeiT-B* [39]	121	2772	44.09	45.68
UperNet	Swin-S† [26]	81	1038	47.60	48.87
UperNet	**S3-S(Ours)**	81	1071	**48.04**	**49.31**

Table 4: Results on VQA v2.0 dataset.

Backbone	#Param (M)	FLOPs (G)	VQA test-dev(%)	VQA test-std(%)
ResNet-50 [13]	4.1	26	64.65	65.12
ResNet-101 [13]	7.8	45	65.44	65.71
Swin-T [26]	4.5	29	67.13	67.40
S3-T (Ours)	4.6	29	**67.84**	**68.00**

Table 5: Comparisons of searched architectures in different spaces.

Space	#Param(M)	FLOPs(G)	Top-1(%)
A	28M	4.7	65.5
B	29M	4.7	78.0
C	28M	4.7	82.1

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 [52] for different vision backbones. Our model S3-T brings more than +2% accuracy gains over ResNet-101 [13], and surpassing Swin-T [26] 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, 53, 5]. ViT [9] is the first one that introduces a pure transformer architecture for visual recognition. Alleviating the data hungry and computational issues, DeiT [39] 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 [43, 14, 30]. Swin Transformer [26] 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 [25, 6, 47] and MBConv-based space [3, 31, 49], is pre-defined and fixed during search. Several recent works [16, 28, 29, 32, 24, 22] perform search space shrinking to find a better compact one. Moreover, NSENet [7] proposes an evolving search space algorithm for MBConv space optimization. RegNet [34] 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 [10, 19]. Early approaches use either reinforcement learning [55, 56] or evolution algorithms [45, 35]. 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 [21, 33, 1, 11]. 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 [36] searches the inner architecture of transformer block. HAT [42] proposes to design hardware-aware transformers with NAS to enable low-latency inference on resource-constrained hardware platforms. BossNAS [20] explores hybrid CNN-Transformers with block-wisely self-supervised. Unlike the above studies, we focus on the search of pure vision transformer architectures and exploring the search space.

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.

References

[1] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. In *ICLR*, 2018. 9

[2] 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. 1

[3] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *ICLR*, 2018. 9

[4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 9

[5] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International Conference on Machine Learning*, pages 1691–1703. PMLR, 2020. 9

[6] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019. 9

[7] Yuanzheng Ci, Chen Lin, Ming Sun, Boyu Chen, Hongwen Zhang, and Wanli Ouyang. Evolving search space for neural architecture search. *arXiv preprint arXiv:2011.10904*, 2020. 9

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 7

[9] 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

[10] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019. 9

[11] 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. *ECCV*, 2020. 1, 3, 4, 9

[12] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on visual transformer. *arXiv preprint arXiv:2012.12556*, 2020. 1

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5, 7, 8, 9

[14] Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers. *arXiv preprint arXiv:2103.16302*, 2021. 1, 9

[15] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, 2019. 1

[16] Yiming Hu, Yuding Liang, Zichao Guo, Ruosi Wan, Xiangyu Zhang, Yichen Wei, Qingyi Gu, and Jian Sun. Angle-based search space shrinking for neural architecture search. *ECCV*, 2020. 9

[17] Zhicheng Huang, Zhaoyang Zeng, Yupan Huang, Bei Liu, Dongmei Fu, and Jianlong Fu. Seeing out of the box: End-to-end pre-training for vision-language representation learning. In *CVPR*, 2021. 8

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

[19] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *arXiv preprint arXiv:2101.01169*, 2021. 1, 9

[20] 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. *arXiv preprint arXiv:2103.12424*, 2021. 9

[21] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *UAI*, 2019. 9

[22] 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. 9

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 8

[24] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018. 9

[25] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019. 9

[26] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021. 1, 2, 3, 7, 8, 9

[27] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 6, 7

[28] Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik. Xnas: Neural architecture search with expert advice. In *NeurIPS*, 2019. 9

[29] Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. Asap: Architecture search, anneal and prune. In *AISTATS*, pages 493–503. PMLR, 2020. 9

[30] Zizheng Pan, Bohan Zhuang, Jing Liu, Haoyu He, and Jianfei Cai. Scalable visual transformers with hierarchical pooling. *arXiv preprint arXiv:2103.10619*, 2021. 9

[31] Houwen Peng, Hao Du, Hongyuan Yu, Qi Li, Jing Liao, and Jianlong Fu. Cream of the crop: Distilling prioritized paths for one-shot neural architecture search. *NeurIPS*, 2020. 9

[32] Juan-Manuel Pérez-Rúa, Moez Baccouche, and Stephane Pateux. Efficient progressive neural architecture search. In *BMVC*, 2018. 9

[33] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *ICML*, 2018. 3, 9

[34] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, 2020. 1, 3, 7, 8, 9

[35] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 9

[36] David So, Quoc Le, and Chen Liang. The evolved transformer. In *International Conference on Machine Learning*. PMLR, 2019. 9

[37] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. *arXiv preprint arXiv:2101.11605*, 2021. 7

[38] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 1, 5, 7, 8

[39] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herv'e J'egou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020. 1, 6, 7, 8, 9

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 1, 6

[41] Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. Attentivenas: Improving neural architecture search via attentive sampling. In *CVPR*, 2021. 1

[42] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. Hat: Hardware-aware transformers for efficient natural language processing. In *ACL*, 2020. 3, 9

[43] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021. 8, 9

[44] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019. 1

[45] Lingxi Xie and Alan Yuille. Genetic cnn. In *ICCV*, 2017. 9

[46] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 8

[47] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: Partial channel connections for memory-efficient architecture search. In *ICLR*, 2020. 9

[48] Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. Nas evaluation is frustratingly hard. In *ICLR*, 2020. 1

[49] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. Greedynas: Towards fast one-shot nas with greedy supernet. In *CVPR*, 2020. 9

[50] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. *NeurIPS*, 2020. 3, 4

[51] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. *arXiv preprint arXiv:2101.11986*, 2021. 1, 7

[52] Peng Zhang, Yash Goyal, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Yin and Yang: Balancing and answering binary visual questions. In *CVPR*, 2016. 8, 9

[53] 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. *arXiv preprint arXiv:2012.15840*, 2020. 9

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

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

[56] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 1, 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]