AutoFormer: Searching Transformers for Visual Recognition

Minghao Chen\textsuperscript{1,*}, Houwen Peng\textsuperscript{2,*†}, Jianlong Fu\textsuperscript{2}, Haibin Ling\textsuperscript{1}
\textsuperscript{1}Stony Brook University \textsuperscript{2}Microsoft Research Asia

Abstract

Recently, pure transformer-based models have shown great potentials for vision tasks such as image classification and detection. However, the design of transformer networks is challenging. It has been observed that the depth, embedding dimension, and number of heads can largely affect the performance of vision transformers. Previous models configure these dimensions based upon manual crafting. In this work, we propose a new one-shot architecture search framework, namely AutoFormer, dedicated to vision transformer search. AutoFormer entangles the weights of different blocks in the same layers during supernet training. Benefiting from the strategy, the trained supernet allows thousands of subnets to be very well-trained. Specifically, the performance of these subnets with weights inherited from the supernet is comparable to those retrained from scratch. Besides, the searched models, which we refer to AutoFormers, surpass the recent state-of-the-arts such as DeiT \cite{tian2021deit} and ResNet \cite{he2016deep}.

In this work, we present a new architecture search algorithm, named AutoFormer, dedicated to finding pure vision transformer models. Our approach mainly addresses two challenges in transformer search. 1) How to strike a good depth, embedding dimension and/or head number? These factors are all critical for elevating model capacity, yet finding a good combination of them is difficult. As seen in Fig. 2, increasing the depth, head number and MLP ratio (the ratio of hidden dimension to the embedding dimension in the multi-layer perceptron) of transformers helps achieve higher accuracy at first but get overfit after hitting the peak value. Scaling up the embedding dimension can improve model capability, but the accuracy gain plateaus for larger models. These phenomenons demonstrate the challenge of designing optimal transformer architectures.

Previous works on designing vision transformers are based upon manual crafting, which heavily relies on human expertise and typically requires a deal of trial-and-error \cite{dosovitskiy2020image,tian2021deit,he2016deep}. There are a few works on automating transformer design using neural architecture search (NAS) \cite{pmlr-v97-falkner19a,pham2018efficient}. However, they are all concentrated on natural language tasks, such as machine translation, which are quite different from computer vision tasks. As a result, it is hard to generalize prior automatic search algorithms to find effective vision transformer architectures.

In this work, we present a new architecture search algorithm, named AutoFormer, dedicated to finding pure vision transformer models. Our approach mainly addresses two challenges in transformer search. 1) How to strike a good
This strategy is different from most one-shot NAS methods where different blocks are maximally entangled during training. We give a detailed discussion in Section 3.4 exploring the underlying reasons of weight entanglement. How to efficiently find out various transformer models that fit different resource constraints and application scenarios? To tackle the challenges, we construct a large search space covering the main changeable dimensions of transformer, including embedding dimension, number of heads, query/key/value dimension, MLP ratio, and network depth. This space contains a vast number of transformers with diverse structures and model complexities. In particular, it allows the construction of transformers to use different structures of building blocks, thus breaking the convention that all blocks share an identical structure in transformer design.

To address the efficiency issue, inspired by BigNAS [65] and slimmable networks [65, 66], we propose a supernet training strategy called weight entanglement dedicated to transformer architecture. The central idea is to enable different transformer blocks to share weights for their common parts in each layer. An update of weights in one block will affect all other ones as a whole, such that the weights of different blocks are maximally entangled during training. This strategy is different from most one-shot NAS methods [16, 8, 59], in which the weights of different blocks are independent for the same layer, as visualized in Fig. 5.

We propose a supernet architecture to incorporate different architectures as its subnets, rather than fixed structures. We perform a evolutionary search with a model size constraint over the well-trained supernets to find promising Transformers. Experiments on ImageNet [11] demonstrate that our method achieves superior performance to the hand-crafted state-of-the-art transformer models. For instance, as shown in Fig. 1, with 22.9M parameters, Autoformer achieves a top-1 accuracy of 81.7%, being 1.8% and 2.9% better than DeiT-S [50] and ViT-S/16 [13], respectively. In addition, when transferred to downstream vision classification datasets, our AutoFormer also performs well with fewer parameters, achieving better or comparable results to the best convolutional models, such as EfficientNet [49].

In summary, we make three major contributions in this paper. 1) To our best knowledge, this work is the first effort to design an automatic search algorithm for finding vision transformer models. 2) We propose a simple yet effective framework for efficient training of transformer supernets. Without extra finetuning or retraining, the trained supernet is able to produce thousands of high quality transformers by inheriting weights from it directly. Such merit allows our method to search diverse models to fit different resource constraints. 3) Our searched models, i.e., AutoFormers, achieve the state-of-the-art results on ImageNet among the vision transformers, and demonstrate promising transferability on downstream tasks.

2. Background

Before presenting our method, we first briefly review the background of the vision transformer and one-shot NAS.

2.1. Vision Transformer

Transformer is originally designed for natural language tasks [52, 28, 12]. Recent works, such as ViT and DeiT [13, 50], show its great potential for visual recognition. In the following, we give the basic pipeline of the vision transformer, which serves as a base architecture of AutoFormer.

Given a 2D image, we first uniformly split it into a sequence of 2D patches just like tokens in natural language processing. We then flatten and transform the patches to D-dimension vectors, named patch embeddings, by either linear projection [13] or several CNN layers [67]. A learnable [class] embedding is injected into the head of the sequence to represent the whole picture. Position embeddings are added to the patch embeddings to retain positional information. The combined embeddings are then fed to a transformer encoder described below. At last, a linear layer is

Figure 2. Adjust a baseline model with different embedding dimension (e), depth (d), MLP ratio (r), and number of heads (h) coefficients under the same training recipe, where MLP ratio (the ratio of hidden dimension to the embedding dimension in the multi-layer perceptron). We set the baseline model with d = 12, r = 4, e = 384, h = 6. Note: number of heads does not affect the model size and complexity if we fix the Q-K-V dimension.
used for the final classification.

A transformer encoder consists of alternating blocks of multihead self-attention (MSA) and multi-layer perceptron (MLP) blocks. LayerNorm (LN) [2] is applied before every block, and residual connections after every block. The details of MSA and MLP are given below.

**Multihead Self-Attention (MSA).** In a standard self-attention module, the input sequence $z \in \mathbb{R}^{N \times D}$ will be first linearly transformed to queries $Q \in \mathbb{R}^{N \times D_h}$, keys $K \in \mathbb{R}^{N \times D_h}$, and values $V \in \mathbb{R}^{N \times D_h}$, where $N$ is the number of tokens, $D$ is the embedding dimension, $D_h$ is the $Q$-$K$-$V$ dimension. Then we compute the weighted sum over all values for each element in the sequence. The weights or attention are based on the pairwise similarity between two elements of the sequence:

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_h}} \right) V,$$

where $\frac{1}{\sqrt{d_h}}$ is the scaling factor. Lastly, a fully connected layer is applied. Multihead self-attention splits the queries, keys and values into different heads and performs self-attention in parallel and projects their concatenated outputs.

**Multi-Layer Perceptron (MLP).** The MLP block consists of two fully connected layers with an activation function, usually GELU [19]. In this work, we focus on finding optimal choices of the MLP ratios in each layer.

### 2.2. One-Shot NAS

One-shot NAS typically adopts a weight sharing strategy to avoid training each subnet from scratch [16, 39]. The architecture search space $\mathcal{A}$ is encoded in a supernet, denoted as $\mathcal{N}(\mathcal{A}, W)$, where $W$ is the weight of the supernet. $W$ is shared across all the architecture candidates, i.e., subnets $\alpha \in \mathcal{A}$ in $\mathcal{N}$. The search of the optimal architecture $\alpha^*$ in one-shot NAS is usually formulated as a two-stage optimization problem. The first-stage is to optimize the weight $W$ by

$$W_\mathcal{A} = \arg \min_W \mathcal{L}_\text{train}(\mathcal{N}(\mathcal{A}, W)),$$

where $\mathcal{L}_\text{train}$ represents the loss function on the training dataset. To reduce memory usage, one-shot methods usually sample subnets from $\mathcal{N}$ for optimization. The second-stage is to search architectures via ranking the performance of subnets $\alpha \in \mathcal{A}$ based on the learned weights in $W_\mathcal{A}$:

$$\alpha^* = \arg \max_{\alpha \in \mathcal{A}} \text{Acc}_{\text{val}}(\mathcal{N}(\alpha, w)),$$

where the sampled subnet $\alpha$ inherits weight $w$ from $W_\mathcal{A}$, and $\text{Acc}_{\text{val}}$ indicates the top-1 accuracy of the architecture $\alpha$ on the validation dataset. Since it is impossible to enumerate all the architectures $\alpha \in \mathcal{A}$ for evaluation, prior works resort to random search [34, 4], evolution algorithms [43, 16] or reinforcement learning [40, 48] to find the most promising one.

### 3. AutoFormer

In this section, we first demonstrate that it is impractical to directly apply one-shot NAS for transformer search following classical weight sharing strategy [16], using different weights for different blocks in each layer, because of the slow convergence and unsatisfactory performance. Then we propose the weight entanglement strategy for vision transformer to address the issues. Finally, we present the search space and search pipeline.
cretely, for a subnet we propose the weight entanglement training strategy proposed by BigNAS [65] and slimmable networks [66, 64], architectures since the weights are not fully optimized. In- necessary to perform additional retraining for the searched pacities of supernet. Furthermore, after the search, it is still (see the right part of Fig.4). This limits the ranking ca-
ances of subnets inheriting weights from the one-shot su-
pernet, trained under classical weight sharing strategy, are
mproved by limited times. 2) Low performance. The perfor-
training of transformer blocks results in the weights being
shown in the Fig.4 (left), the training loss of the supernet
49]. However, in transformer search space, this classical
par with those of training from scratch. 

3.1. One-Shot NAS with Weight Entanglement

Prior one-shot NAS methods commonly share weights across architectures during supernet training while decoupling the weights of different operators at the same layer. Such strategy performs well when used to search architectures over convolutional neural networks space [16, 6, 8, 21, 49]. However, in transformer search space, this classical strategy encounters difficulties. 1) Slow convergence. As shown in the Fig. 4 (left), the training loss of the supernet converges slowly. The reason might be that the independent training of transformer blocks results in the weights being updated by limited times. 2) Low performance. The performances of subnets inheriting weights from the one-shot supernet, trained under classical weight sharing strategy, are far below their true performances of training from scratch (see the right part of Fig. 4). This limits the ranking capacities of supernet. Furthermore, after the search, it is still necessary to perform additional retraining for the searched architectures since the weights are not fully optimized. In- spiried by BigNAS [65] and slimmable networks [66, 64], we propose the weight entanglement training strategy dedicated to vision transformer architecture search. The central idea is to enable different transformer blocks to share weights for their common parts in each layer. More concretely, for a subnet $\alpha \in \mathcal{A}$ with a stack of $l$ layers, we represent its structure and weights as

$$
\begin{align*}
\alpha &= (\alpha^{(1)}, ... \alpha^{(i)}, ... \alpha^{(l)}), \\
\omega &= (\omega^{(1)}, ... \omega^{(i)}, ... \omega^{(l)}),
\end{align*}
$$

where $\alpha^{(i)}$ denotes the sampled block in the $i$-th layer and $\omega^{(i)}$ is the block weights. During architecture search, there are multiple choices of blocks in each layer. Hence, $\alpha^{(i)}$ and $\omega^{(i)}$ are actually selected from a set of $n$ block candidates belonging to the search space, which is formulated as

$$
\begin{align*}
\alpha^{(i)} &\in \{b^{(i)}_1, ... b^{(i)}_j, ... b^{(i)}_n\}, \\
\omega^{(i)} &\in \{w^{(i)}_1, ... w^{(i)}_j, ... w^{(i)}_n\},
\end{align*}
$$

where $b^{(i)}_j$ is a candidate block in the search space and $w^{(i)}_j$ is its weights. The weight entanglement strategy enforces that different candidate blocks in the same layer to share as many weights as possible. This requires that, for any two blocks $b^{(i)}_j$ and $b^{(i)}_k$ in the same layer, we have

$$
W^{(i)}_j \subseteq W^{(i)}_k \quad \text{or} \quad W^{(i)}_k \subseteq W^{(i)}_j.
$$

Such within layer weight sharing makes the weight updates of $W^{(i)}_j$ and $W^{(i)}_k$ entangled with each other. The training of any block will affect the weights of others for their inter-
sected portion, as demonstrated in Fig. 5. This is different from the classical weight sharing strategy in one-shot NAS, where the building blocks in the same layer are isolated. In other words, in classical weight sharing, for any two blocks $b^{(i)}_j$ and $b^{(i)}_k$, we have $W^{(i)}_j \cap W^{(i)}_k = \emptyset$.

Note that the proposed weight entanglement strategy is dedicated to work on homogeneous building blocks, such as self-attention modules with different numbers of heads, and multi-layer perceptron with different hidden dimensions. The underlying reason is that homogeneous blocks are structurally compatible, such that the weights can share with each other. During implementation, for each layer, we need to store only the weights of the largest block among the $n$ homogeneous candidates. The remaining smaller building blocks can directly extract weights from the largest one.

Equipped with weight entanglement, one-shot NAS is capable of searching transformer architectures in an efficient and effective fashion, as demonstrated in Fig. 4.

Compared with classical weight sharing methods, our weight entanglement strategy has three advantages. 1) Faster convergence. Weight entanglement allows each block to be updated more times than the previous independent training strategy. 2) Low memory cost. We now only need to store the largest building blocks’ parameters for each layer, instead of all the candidates in the space. 3) Better subnets performance. We found that the subnets trained with weight entanglement could achieve performance on par with those of training from scratch.

3.2. Search Space

We design a large transformer search space that includes five variable factors in transformer building blocks: embedding dimension, $Q$-$K$-$V$ dimension, number of heads, MLP ratio, and network depth, as detailed in Tab. 1 and Fig. 3. All
these factors are important for model capacities. For example, in attention layers, different heads are used to capture various dependencies. However, recent works [36, 53, 9] show that many heads are redundant. We thereby make the attention head number elastic so that each attention module can decide its necessary number of heads. On the other hand, different layers have different capacities on feature representation, the varying hidden dimensions in layers might be better than the fixed sizes when used for constructing new models. Moreover, AutoFormer adds new Q-K-V dimension into the search space and fixes the ratio of the Q-K-V dimension to the number of heads in each block. This setting makes the scaling factor $\frac{1}{\sqrt{d_k}}$ in attention calculation invariant to the number of heads, stabilizing the gradients, and decouples the meaning of different heads.

Following one-shot NAS methods, we encode the search space into a supernet. That is, every model in the supernet is a part/subset of the supernet. All subnets share the weights of their common parts. The supernet is the largest model in the space, and its architecture is shown in Fig. 3. In particular, the supernet stacks the maximum number of transformer blocks with the largest embedding dimension, Q-K-V dimension and MLP ratio as defined in the space. During training, all possible subnets are uniformly sampled, and the corresponding weights are updated.

According to the constraints on model parameters, we partition the large-scale search space in to three parts and encode them into three independent supernets, as elaborated in Tab. 1. Such partition allows the search algorithm to concentrate on finding models within a specific parameter range, which can be specialized by users according to their available resources and application requirements.

Overall, our supernets contain more than $1.7 \times 10^{16}$ candidate architectures covering a wide range of model size.

### 3.3. Search Pipeline

Our search pipeline includes two sequential phases.

**Phase 1: Supernet Training with Weight Entanglement.** In each training iteration, we uniformly sample a subnet $\alpha = (\alpha^{(1)}_1, ..., \alpha^{(i)}_1, ..., \alpha^{(l)}_1)$ from the per-defined search space and update its corresponding weights $w = (w^{(1)}_1, ..., w^{(i)}_1, ..., w^{(l)}_1)$ in the supernet’s weight $W_A$ while freezing the rest. Detailed algorithm is given in supplementary materials, Appendix A.

**Phase 2: Evolution Search under Resource Constraints.** After obtaining the trained supernet, we perform an evolution search on it to obtain the optimal subnets. Subnets are evaluated and picked according to the manager of the evolution algorithm. Our objective here is to maximize the classification accuracy while minimizing the model size. At the beginning of the evolution search, we pick $N$ random architecture as seeds. The top $k$ architectures are picked as parents to generate the next generation by crossover and mutation. For a crossover, two randomly selected candidates are picked and crossed to produce a new one during each generation. For mutation, a candidate mutates its depth with probability $P_d$ first. Then it mutates each block with a probability of $P_m$ to produce a new architecture.

### 3.4. Discussion

**Why does weight entanglement work?** We conjecture that there are two underlying reasons. 1) Regularization in training. Different from convolution neural networks, transformer has no convolution operations at all. Its two basic components, MSA and MLP, employ only fully connected layers. Weight entanglement could be viewed a regularization training strategy for transformer, to some extent, similar to the effects of dropout [47, 54, 30]. When sampling the small subnets, corresponding units cannot rely on other hidden units for classification, which hence reduces the reliance of units. 2) Optimization of deep thin subnets. Recent works [3, 57] show that deep transformer is hard to train, which coincides with our observation in Fig. 2. This is because the gradients might explode or vanish in deep thin networks during backpropagation. Increasing the width of or “overparameterizing” the network will help the optimization [33, 14, 1, 71]. Our weight entanglement training strategy helps to optimize the thin subnets in a similar way. The gradients backwared by wide subnets will help to update the weights of thin subnets. Besides, the elastic depth severs similar effects to stochastics depth [23] and deep supervision [29], which supervise the shallow layers as well.

### 4. Experiments

In this section, we first present the implementation details and evolution search settings. We then analyze the proposed weight entanglement strategy and provide a large number of well-trained subnets sampled from supernets to demonstrate its efficacy. At last, we present the performance of AutoFormer evaluated on several benchmarks with comparisons with state-of-the-art models designed manually or automatically.
Table 2. Comparison of different search methods. The supernets are trained for 500 epochs, while the subnets are retrained for 300 epochs. Random search are performed three times and the best performance is reported.

<table>
<thead>
<tr>
<th>Search Method</th>
<th>Inherited</th>
<th>Retrain</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Search</td>
<td></td>
<td></td>
<td>23.0M</td>
</tr>
<tr>
<td>Classical Weight Sharing + Random Search</td>
<td>69.7%</td>
<td>80.1%</td>
<td>22.9M</td>
</tr>
<tr>
<td>Weight Entanglement + Random Search</td>
<td>81.3%</td>
<td>81.4%</td>
<td>22.8M</td>
</tr>
<tr>
<td>Classical Weight Sharing + Evolution Search (SPOS [16])</td>
<td>71.5%</td>
<td>80.4%</td>
<td>22.9M</td>
</tr>
<tr>
<td>Weight Entanglement + Evolution Search (Ours)</td>
<td>81.7%</td>
<td>81.7%</td>
<td>22.9M</td>
</tr>
</tbody>
</table>

Table 3. Supernet training settings. LR refers to learning rate.

<table>
<thead>
<tr>
<th>Epochs</th>
<th>Optimizer</th>
<th>Batch Size</th>
<th>LR</th>
<th>LR scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>AdamW</td>
<td>1024</td>
<td>1e-3</td>
<td>cosine</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weight Decay</th>
<th>Warmup Epochs</th>
<th>Label Drop</th>
<th>Repeated Smoothing</th>
<th>Path Augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5e-2</td>
<td>20</td>
<td>0.1</td>
<td>0.1</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 4. Comparison of subnets with inherited weights, fine-tuned (40 epochs) and trained from-scratch (300 epochs).

<table>
<thead>
<tr>
<th>Model</th>
<th>Model Size</th>
<th>Inherited</th>
<th>Finetune</th>
<th>Retrain</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoFormer-T</td>
<td>5.7M</td>
<td>74.7%</td>
<td>74.9%</td>
<td>74.9%</td>
</tr>
<tr>
<td>AutoFormer-S</td>
<td>22.9M</td>
<td>81.7%</td>
<td>81.8%</td>
<td>81.7%</td>
</tr>
<tr>
<td>AutoFormer-B</td>
<td>53.7M</td>
<td>82.4%</td>
<td>82.6%</td>
<td>82.6%</td>
</tr>
</tbody>
</table>

4.1. Implementation Details

**Supernet Training.** We train the supernets using a similar recipe as DeiT [50]. The details are presented in Tab. 3. Data augmentation techniques, including RandAugment [10], Cutmix [68], Mixup [69] and random erasing, are adopted with the same hyperparameters as in DeiT [50] except the repeated augmentation [20]. Images are split into patches of size 16x16. All the models are implemented using PyTorch 1.7 and trained on Nvidia Tesla V100 GPUs.

**Evolutionary Search.** The implementation of evolution search follows the same protocol as in SPOS [16]. For a fair comparison, we reserve the ImageNet validation set for testing and subsample 10,000 training examples (100 images per class) as the validation dataset. We set the population size to 50 and number of generations to 20. Each generation we pick the top 10 architectures as the parents to generate child networks by mutation and crossover. The mutation probability $P_d$ and $P_m$ are set to 0.2 and 0.4.

4.2. Ablation Study and Analysis

**The Efficacy of Weight Entanglement.** We compare AutoFormer with random search and SPOS [16] (classical weight sharing) baselines to demonstrate the effectiveness of weight entanglement. For random search, we randomly pick up architectures from the search space to meet the model size constraints. For SPOS [16], we adapt it to the transformer search space defined in Tab. 1 and keep the remaining configurations to be consistent with the original method. In other words, in each layer of the SPOS supernet, the transformer blocks with different architecture arguments do not share weights. For example, in an MLP block, there are multiple choices of hidden dimensions. Each MLP choice has its own weights, being independent of each other. After training the SPOS supernet, we apply the same evolution process to find the most promising architecture candidate and retrain it using the same setting as our AutoFormer.

Table 2 presents the comparisons on ImageNet. We can observe that: 1) After retraining, random search and SPOS are 2.3% and 1.3% inferior to our methods indicating the superiority of the proposed methods. 2) Without retraining, i.e., inheriting weights directly from the supernet, the weight entanglement training strategy can produce significantly better results than the classical weight sharing. The entangled weight can produce well-trained subnets, which are very close to the ones retrained from scratch. We conjecture the relatively inferior performance of SPOS in transformer space is mainly due to insufficient training. We also observe that if we train the supernet in SPOS for more epochs, the performance can be slowly improved. However, its training cost is largely higher than our proposed weight entanglement strategy. Fig. 6 plots the accuracy over the number of architectures sampled from the trained supernet during search. Top 50 candidates are depicted at each generation. It is clear that the evolution search on the supernet is more effective than the random search baseline.
Table 5. AutoFormer performance on ImageNet with comparisons to state-of-the-arts. We group the models according to their parameter sizes. Our AutoFormer consistently outperforms existing transformer-based visual models, being comparable to CNN models.†: reported by [50], ⋆: reported by [63].

<table>
<thead>
<tr>
<th>Models</th>
<th>Top-1 Acc.</th>
<th>Top-5 Acc.</th>
<th>#Parameters</th>
<th>FLOPs</th>
<th>Resolution</th>
<th>Model Type</th>
<th>Design Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV3$	ext{Large}$.0 [21]</td>
<td>75.2%</td>
<td>-</td>
<td>5.4M</td>
<td>0.22G</td>
<td>224²</td>
<td>CNN</td>
<td>Auto</td>
</tr>
<tr>
<td>EfficientNet-B0[49]</td>
<td>77.1%</td>
<td>93.3%</td>
<td>5.4M</td>
<td>0.39G</td>
<td>224²</td>
<td>CNN</td>
<td>Auto</td>
</tr>
<tr>
<td>DeiT-tiny [50]</td>
<td>72.2%</td>
<td>91.1%</td>
<td>5.7M</td>
<td>1.2G</td>
<td>224²</td>
<td>Transformer</td>
<td>Manual</td>
</tr>
<tr>
<td><strong>AutoFormer-tiny (Ours)</strong></td>
<td><strong>74.7%</strong></td>
<td><strong>92.6%</strong></td>
<td><strong>5.7M</strong></td>
<td><strong>1.3G</strong></td>
<td><strong>224²</strong></td>
<td><strong>Transformer</strong></td>
<td><strong>Auto</strong></td>
</tr>
<tr>
<td>ResNet50* [18]</td>
<td>79.1%</td>
<td>-</td>
<td>25.5M</td>
<td>4.1G</td>
<td>224²</td>
<td>CNN</td>
<td>Manual</td>
</tr>
<tr>
<td>RegNet-Y-4GF[41]</td>
<td>80.0%</td>
<td>-</td>
<td>21.4M</td>
<td>4.0G</td>
<td>224²</td>
<td>CNN</td>
<td>Auto</td>
</tr>
<tr>
<td>EfficientNet-B4 [49]</td>
<td>82.9%</td>
<td>95.7%</td>
<td>19.3M</td>
<td>4.2G</td>
<td>380²</td>
<td>CNN</td>
<td>Auto</td>
</tr>
<tr>
<td>BoTNet-S1-S9 [46]</td>
<td>81.7%</td>
<td>95.8%</td>
<td>33.5M</td>
<td>7.3G</td>
<td>224²</td>
<td>CNN + Trans</td>
<td>Manual</td>
</tr>
<tr>
<td>T2T-ViT-I4 [67]</td>
<td>81.7%</td>
<td>-</td>
<td>21.5M</td>
<td>6.1G</td>
<td>224²</td>
<td>Transformer</td>
<td>Manual</td>
</tr>
<tr>
<td>DeiT-S [50]</td>
<td>79.9%</td>
<td>95.0%</td>
<td>22.1M</td>
<td>4.7G</td>
<td>224²</td>
<td>Transformer</td>
<td>Manual</td>
</tr>
<tr>
<td>ViT-S/16 [13]</td>
<td>78.8%</td>
<td>-</td>
<td>22.1M</td>
<td>4.7G</td>
<td>384²</td>
<td>Transformer</td>
<td>Manual</td>
</tr>
<tr>
<td><strong>AutoFormer-small (Ours)</strong></td>
<td><strong>81.7%</strong></td>
<td><strong>95.7%</strong></td>
<td><strong>22.9M</strong></td>
<td><strong>5.1G</strong></td>
<td><strong>224²</strong></td>
<td><strong>Transformer</strong></td>
<td><strong>Auto</strong></td>
</tr>
<tr>
<td>ResNet152* [18]</td>
<td>80.8%</td>
<td>-</td>
<td>60M</td>
<td>11G</td>
<td>224²</td>
<td>CNN</td>
<td>Manual</td>
</tr>
<tr>
<td>EfficientNet-B7 [49]</td>
<td>84.3%</td>
<td>97.0%</td>
<td>66M</td>
<td>37G</td>
<td>600²</td>
<td>CNN</td>
<td>Auto</td>
</tr>
<tr>
<td>DeiT-B [50]</td>
<td>81.8%</td>
<td>95.6%</td>
<td>86M</td>
<td>18G</td>
<td>224²</td>
<td>Transformer</td>
<td>Manual</td>
</tr>
<tr>
<td><strong>AutoFormer-base (Ours)</strong></td>
<td><strong>82.4%</strong></td>
<td><strong>95.7%</strong></td>
<td><strong>54M</strong></td>
<td><strong>11G</strong></td>
<td><strong>224²</strong></td>
<td><strong>Transformer</strong></td>
<td><strong>Auto</strong></td>
</tr>
</tbody>
</table>

Subnet Performance without Retraining. We surprisingly observe that there are a large number of subnets achieving very good performance when inheriting weights from the supernets, without extra finetuning or retraining. The blue points shown in Fig. 7 represents the 1000 high-performing subsets sampled from the supernet-S. All these subsets can achieve top-1 accuracies ranging from 80.1% to 82.0%, exceeding the recent DeiT [50] and RegNetY [41]. Such results amply demonstrate the effectiveness of the proposed weight entanglement strategy for one-shot supernet training. Tab. 4 shows that if we further finetune or retrain the searched subnets on ImageNet, the performance gains are very small, even negligible. This phenomenon illustrates the weight entanglement strategy allows the subsets to be well-trained in supernets, leading to the facts that searched transformers do not require any retraining or finetuning and the supernet itself serves good indicator of subnets’ ranking.

4.3. Results on ImageNet

We perform the search of AutoFormer on ImageNet and find multiple transformer models with diverse parameter sizes. All these models inherit weights from supernets directly, without extra retraining and other postprocessing. The performance are reported in Tab. 5 and Fig. 1. It is clear that our AutoFormer model families achieve higher accuracies than the recent handcrafted state-of-the-art transformer models such as ViT [13] and DeiT [50]. In particular, using ~23M parameters, our small model, i.e. AutoFormer-S, achieves a top-1 accuracy of 81.7%, being 1.8% and 2.9% better than DeiT-S and ViT-S/16, respectively.

Compared to vanilla CNN models, AutoFormer is also competitive. As visualized in Fig. 1, our AutoFormers perform better than the manually-designed ResNet [18], ResNeXt [62] and DenseNet [22], demonstrating the potentials of pure transformer models for visual representation.

However, transformer-based vision models, including AutoFormer, now are still inferior to the models based on inverted residual blocks [44], such as MobileNetV3 [21] and EfficientNet [49]. The reason is that inverted residuals are optimized for edge devices, so the model sizes and FLOPs are much smaller than vision transformers.

4.4. Transfer Learning Results

Classification. We transfer Autoformer to a list of commonly used recognition datasets: 1) general classification: CIFAR-10 and CIFAR-100 [27]; 2) fine-grained classification: Stanford Car [26], FFlowers [37] and Oxford-III Pets [38]. We follow the same training settings as DeiT [50], which take ImageNet pretrained checkpoints and finetune on new datasets. Tab. 6 shows the results in terms of top-1 accuracy: 1) Compared to state-of-the-art ConvNets, AutoFormer is close to the best results with a negligible gap with fewer parameters; 2) Compared to transformer-based models, AutoFormer achieves better or comparable results on all datasets, with much fewer parameters (~4x).

Distillation. AutoFormer is also orthogonal to knowledge distillation (KD) since we focus on searching for an efficient architecture while KD focuses on better training a given architecture. Combining KD with AutoFormers by distilling hard labels from a RegNetY-32GF [41] teacher could further improve the performance from 74.7%/81.7%/82.4% to 75.7%/82.4%/82.9%, respectively.
5. Related Work

**Vision Transformer.** Transformer is originally proposed for language modeling [52], and recently applied in computer vision. It has shown promising potentials on a variety of tasks [7, 13, 35]. A straightforward approach for using transformer in vision is to combine convolutional layers with the self-attention module [52, 58]. There has been progress in this direction, such as [42, 70, 56, 24].

Most recently, Dosovitskiy et al. introduce Vision Transformer (ViT) [13], a pure transformer architecture for visual recognition. It presents promising results when trained with an extensive image dataset (JFT-300M, 300 million images) that is not publicly available. The most recent DeiT [50, 67] models verify that large-scale data is not required. Using only ImageNet can also produce a competitive convolution-free transformer. However, existing visions of transformer models are all built upon manual design, which is engineering-expensive and error-prone. In this work, we present the first effort on automating the design of vision transformer with neural architecture search.

**Neural Architecture Search.** There has been an increasing interest in NAS for automating network design [15, 25]. Early approaches search a network using either reinforcement learning [73, 72] or evolution algorithms [61, 43]. Most recent works resort to the one-shot weight sharing strategy to amortize the searching cost [34, 40, 5, 16]. The key idea is to train a over-parameterized supernet model, and then share the weights across subnets. However, most weight-sharing methods need an additional *retraining* step after the search, while the best architecture is identified [16, 31, 59].

Recent works, OFA [6], BigNAS [65] and slimmable networks [66, 64] alleviate this issue by training a once-for-all supernet. Despite the fact that AutoFormer shares similarities with these methods in the concept of training a one-for-all supernet, these methods are designed to search for convolutional networks rather than vision transformers. Specifically, AutoFormer considers the design of multi-head self-attention and MLP, which is unique to transformer models, and gives dedicated design of search dimensions as elaborated in Sec. 3.2. Moreover, BigNAS adopts several well-crafted techniques, such as sandwich training, inplace distillation, regularization, etc. OFA proposes a progressively shrinking approach by progressively distilling the full network to obtain the smaller subnets. By contrast, AutoFormer is simple and efficient, achieving once-for-all training without these techniques.

For transformers, there are few studies applying NAS to improve their architectures [45, 55]. These approaches mainly focus on natural language processing tasks. Among them, the most similar one to us is HAT [55]. In addition to the difference between tasks, HAT requires an additional *retraining or fine-tuning* step after the search, while AutoFormer does not, which is the key difference. Another difference is the search space. HAT searches for an encoder-decoder Transformer structure, while ours is a pure encoder one. There are two concurrent works, *i.e.*, BossNAS [32] and CvT [60], exploring different search space from ours. BossNAS searches for CNN-transformer hybrids, while ours for pure transformers. CvT proposes a new architecture family and searches for the strides and kernel size of them. Due to the difference of search space, we do not compare them in this work.

### 6. Conclusion

In this work, we propose a new one-shot architecture search method, AutoFormer, dedicated to transformer search. AutoFormer is equipped with the training strategy, Weight Entanglement. Under this strategy, the subnets in the search space are almost fully trained. Extensive experiments demonstrate the proposed algorithm can improve the training of supernet and find promising architectures. Our searched AutoFormers achieve state-of-the-art results on ImageNet among vision transformers. Moreover, AutoFormers transfer well to several downstream classification tasks and could be further improved by distillation. In future work, we are interested in further enriching the search space by including convolutions as new candidate operators. Applying weight entanglement to convolution network search or giving the theoretical analysis of the weight entanglement are other potential research directions.

### Acknowledgement

We are very grateful to Xingxing Zhang’s valuable advice and discussion.
References


[26] Jonathan Krause, Jia Deng, Michael Stark, and Li Fei-Fei. Collecting a large-scale dataset of fine-grained cars. 2013. 7


[34] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In UIAI, 2019. 3, 8


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


[54] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In ICCV, 2019.


[72] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. ICLR, 2016. 8