TRIMSFORMER: TRIMMING TRANSFORMER VIA LOW-RANK STRUCTURE SEARCHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision Transformers (ViT) have recently been used successfully in various computer vision tasks, but the high computational cost hinders their practical deployment. One of the most well-known methods to alleviate computational burden is low-rank approximation. However, how to automatically search for a low-rank configuration efficiently remains a challenge. In this paper, we propose Trimsformer, an end-to-end automatic low-rank approximation framework based on a neural architecture search scheme, which tackles the inefficiency of searching for a target low-rank configuration out of numerous ones. We propose *weight inheritance* which encodes enormous rank choices into a single search space. In addition, we share the gradient information among building blocks to boost the convergence of the supernet training. Furthermore, to mitigate the initial performance gap between subnetworks caused by using pre-trained weights, we adopt *non-uniform sampling* to promote the overall subnetwork performance. Extensive results show the efficacy of our Trimsformer framework. For instance, with our method, Trim-DeiT-B/Trim-Swin-B can save up to 57%/46% FLOPs with 1.1%/0.2% higher accuracy over DeiT-B/Swin-B. Last but not least, Trimsformer exhibits remarkable generality and orthogonality. We can yield extra 21%$\sim$26% FLOPs reductions on top of the popular compression method as well as the compact hybrid structure. Our code will be released.

1 INTRODUCTION

The transformer architecture (Vaswani et al., 2017) has dominated natural language processing (NLP) tasks with impressive results. Though intuitively, transformer model seems inept to the special inductive bias of space correlation for images-oriented tasks, it has proved itself of capability on vision tasks just as good as CNN (Dosovitskiy et al., 2021). Since then, vision transformers (ViT) and its variants have shown great potential for image classification (Zhai et al., 2021), object detection (Wei et al., 2022), and semantic segmentation (Liu et al., 2021a). However, the model requires a large number of parameters and high computational cost to obtain higher accuracy, making it unsuitable for edge computing. That is mainly due to the stack of self-attention modules that suffer from quadratic complexity with regard to the input size, among other factors. Hence, research on efficient transformer models has been gaining importance recently.

Earlier works on compressing ViTs mainly follow the techniques for compressing NLP models, ranging from unstructured pruning (Zhu et al., 2021), attention head/structured pruning (Yu et al., 2022b; Chen et al., 2021d), token pruning (Kong et al., 2021; Rao et al., 2021; Yin et al., 2022); to knowledge distillation (Touvron et al., 2021; Jia et al., 2021) and quantization (Yuan et al., 2021; Liu et al., 2021b).

Aside from the above-mentioned directions, another important category of method that employ efficiency in neural network (NN) structure is low-rank approximation. In the case of 2D low-rank approximation, singular-value decomposition (SVD) minimizes the Frobenius norm of the difference between the original matrix and the approximated matrix. Yet, SVD cannot be directly utilized for convolutions in CNNs because weights need to be represented by higher-dimensional (e.g., 4D) tensors (Lee et al., 2019). Special design (Kim et al., 2016; Lebedev et al., 2015) is developed for CNNs by decomposing them into multiple consecutive tensors. However, there is still significant

accuracy drop even after fine-tuning as training is performed on the transformed network structure with consecutive tensors without activation functions in-between. As a result, convergence can be degraded due to vanishing or exploding gradients (Lee et al., 2019). This obstacle is largely alleviated in vision transformer since 90% of total parameters and operations (see Appendix A.1 for detailed analysis) are linear module and conduct matrix multiplication. Linear modules can be decomposed into just two consecutive matrices. To put it in another way, ViTs are much more friendly than CNNs when incorporating low-rank decomposition in the structure. Meanwhile, low-rank decomposition consider redundancy in deep neural networks as noise that contains a very small percentage of variance. Hence, it's different from general pruning methodologies for NN compression, in the way that the overall structural dimension and information flow will not be affected or truncated through low-rank guided compression techniques.

In literature, LRT (Winata et al., 2020) first applied this strategy on transformer in speech recognition. It has been a very effective approach to address the problem of large model size and prohibitive computational cost. However, there has been no systematic study that strives to conduct a low-rank decomposition for vision transformer. Besides, most of the previous methods construct the low-rank structure through manual design. Furthermore, to the best of our knowledge, there has been no sound study on searching for the optimal low-rank architecture of pre-trained transformer model. Therefore, in this paper, we endeavor to find the potential of automatic low-rank ViT to find the optimal trade-off between accuracy and efficiency.

Based on neural architecture search (NAS), we propose Trimsformer, an end-to-end automatic low-rank approximation framework which decomposes linear modules in ViT and searches for rank level to reconstruct vision transformer architecture.

Our main contributions are outlined as follows:

- We set up a first-of-its-kind low-rank architecture search regime for vision transformer inspired by fusing components of SVD decomposition, and design the searching space as the low-rank approximation for the linear components in the architecture.
- To realize efficient and effective supernet training, we 1) design a weight inheritance linear module to enable weight and gradient sharing across different building blocks within the same choice block; 2) select the subnetworks with non-uniform sampling during training to better distribute training resources to subnetworks with different size for balance. Based on our proposed non-uniform sampling and weight inheritance strategy, we enhance the supernet training and achieve outstanding results.
- We conduct a series of experiments to demonstrate Trimsformer's orthogonality to other compression techniques and its generalization capability on ViT variants, which provides a new perspective on high ratio compression for ViT. Extensive experiments further show competitive results on ImageNet-1k when compressing DeiT and Swin-Transformer. For instance, Trim-DeiT-S, Trim-DeiT-B, Trim-Swin-S, and Trim-Swin-B achieve 30%, 58%, 41%, and 46% FLOPs reduction on corresponding backbone with accuracy improvement of 0.6%, 1.1%, 0.2%, and 0.2%, respectively, when compared with the uncompressed model.

2 RELATED WORK

2.1 TRANSFORMER COMPRESSION

Compression methods for transformers can be broadly categorized into: pruning, token reduction and efficient architecture design. Pruning techniques are proposed to alleviate the high computational cost and memory usage by removing the redundant weights in the transformer-based models. VTP (Zhu et al., 2021) reduced the number of embedding dimensions by extending the network slimming approach Liu et al. (2017) to ViTs. Fan et al. (2020); Hou et al. (2020) proposed to skip the inessential layers to obtain a shallow model. Similarly, WDPruning (Yu et al., 2022a) removed the less significant channels of the linear projection by using a neural-network-based saliency predictor. For the token-reduction-based techniques, Tang et al. (2021); Rao et al. (2021) hierarchically remove the redundant patches, thus reducing the computational overhead by slimming the input features.

Aside from above, some other works dedicated on design a efficient architecture directly by introducing CNN to form a hybrid structure. MobileViT (Mehta & Rastegari, 2022) mixed global processing in transformers with convolution, which learns better representations with fewer parameters and simple training recipes.

Recent work endeavored to combine multiple pruning strategies into a unified framework, *i.e.*, considering multiple dimensions simultaneously. For instance, UVC (Yu et al., 2022b) pruned the heads in MHSA, channels in linear projection, and considered layer skipping. Meanwhile, MDC (Hou & Kung, 2022) jointly optimized an extra dimension, the number of patches.

2.2 Low-Rank Approximation

Aside from the model compression technique mentioned in the previous section, the low-rank matrix factorization on weights is also an effective methodology to reduce computational burden. In CNN-based model, Tai et al. (2016) split the convolution kernel into two small ones with fixed rank. Some other work studied on rank selection to generate more fine-grained low-rank approximation. Xu et al. (2020) selected the rank base on thresholding. Idelbayev & Carreira-Perpiñán (2020) introduced rank-based cost function and formulated a constraint optimization problem to decide the rank selection during the training time. Targeting transformer-based model, LRT (Winata et al., 2020) utilized low-rank factorization to approximate the original linear layer with two smaller ones and demonstrate its potential on speech recognition tasks; however, the effect of low-rank approximation on linear layers has not yet been well-studied in ViTs. In this work, we explore low-rank pruning as well as the rank searching on ViT and formulate it as a powerful plug-in.

2.3 One-shot NAS

One-shot NAS (Elsken et al., 2019; Brock et al., 2018; Cai et al., 2019; Stamoulis et al., 2019; Guo et al., 2020) have been proposed to find an efficient architecture based on the two-stage strategy: 1) training an over-parameterized supernet; 2) searching for the optimal subnetwork. SPOS Guo et al. (2020) introduced the evolutionary algorithm to increase searching performance. OFA (Cai et al., 2020) took the hardware constraint into consideration and trained a once-for-all supernet for fast subnetwork selection during inference. These methodologies worked on CNN search space (*e.g.* kernel size, number of channels). Targeting on transformer, HAT (Wang et al., 2020) focused on using the supernet for hardware-aware language transformer. AutoFormer (Chen et al., 2021b) was proposed for ViT by searching for optimal features, such as the number of heads, MLP ratio, embedding dimension, and the number of transformer blocks. GLiT (Chen et al., 2021a) further introduced the locality module and added the CNN correlated features into the search space to decrease the computational cost and explicitly enable local correlation modeling between patches.

3 METHODOLOGY – TRIMSFORMER

Inspired by prior approaches, we propose to solve the low-rank decomposition design guided by an one-shot NAS scheme, which can automatically generate a more fine-grained configuration instead of handcrafting it. To formulate it into an one-shot NAS problem, we need to answer the following question: 1) How to introduce low-rank as a unique and unexplored feature that is orthogonal to existing ones? 2) How to effectively train a supernet initialized with the pre-trained weights?

To answer this question, we formulate an end-to-end framework for automatic compression consisting of three stages: (I) building a supernet with weight inheritance low-rank linear layers as choice blocks (Sec 3.1, 3.2); (II) sampling subnetworks with a non-uniform sampling strategy and training them with soft label distillation (Sec 3.3); and (III) searching for the optimal subnetworks with an evolutionary algorithm (Sec 3.4). Based on our framework, we successfully introduce low-rank approximation into the search space; through weight inheritance and non-uniform sampling, we further support supernet training and acquire efficient low-rank ViTs with competitive performance. The general overview of our framework is summarized in Fig 1.

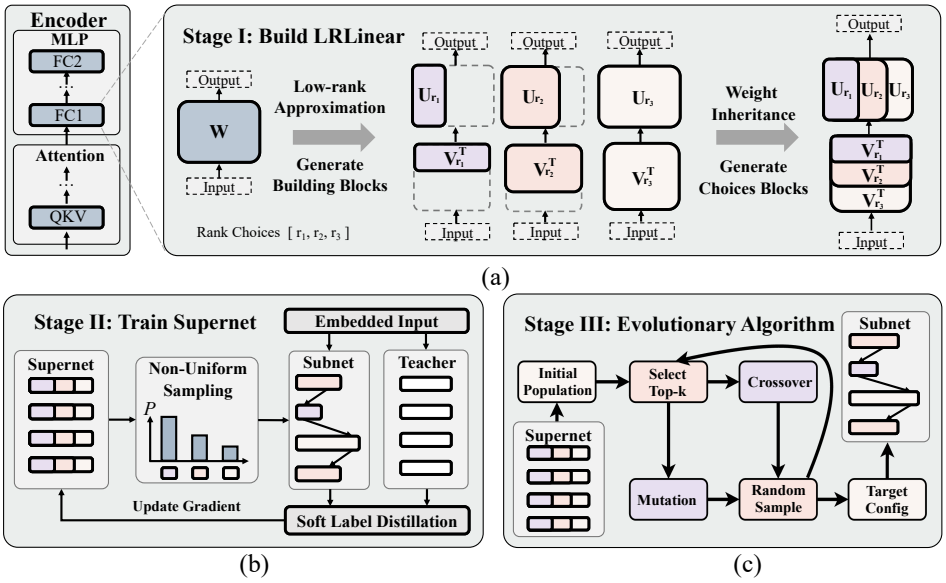

Figure 1: Overview of Trimsformer framework. **(a)**: Workflow to build LRLinear given the original weight matrix and rank choices. The workflow is applied to the linear layers of the MLP and Attention to construct the supernet. **(b)**: Training the supernet with Non-Uniform Sampling (NUS) and soft label distillation. **(c)**: Evolutionary algorithm for optimal architecture searching.

3.1 Encoding Low-Rank into the Search Space

ViT blocks consist of two modules. The main component is a Multi-Head Self Attention (MHSA) module with QKV calculation, follows by three linear layers where the latter two compose of the MLP module. Our main target is to decompose ViT constituent to allow efficient inference.

To introduce the low-rank feature to the transformer, we want to substitute the linear components mentioned above with low-rank approximated linear calculation. After performing low-rank approximation (More details in Appendix A.2), the weight matrix $W \in \mathbb{R}^{m\times n}$ in a linear component is replaced by two small matrices $U_r \in \mathbb{R}^{m\times r}$ and $V_r \in \mathbb{R}^{n\times r}$, where r is set according to the rank under given budget constraint. If we choose a sufficient small $r < \frac{mn}{m+n}$, the number of parameters and FLOPs are reduced from $\mathcal{O}(mn)$ to $\mathcal{O}(rm + rn)$.

Then it is critical to decide the optimal rank configuration for each linear components. To achieve this, we resort to the architecture search scheme with one-shot NAS. Thus, we need to introduce low-rank into the search space. To do so, we treat a low-rank approximated linear component with a specific rank as a building block. We design N candidate rank level choices set $\mathbb{C}$. For each specific rank choice r_i, the building blocks (U_{r_i}, V_{r_i}) are generated. All the proposed building block are gathered into a set $\mathbb{S}$, serving as a choice block. Without loss of generality, we assume that $r_1 < r_2 < ... < r_N$. Thus, for each linear component, we have rank level choices $\mathbb{C}$ and choice block set $\mathbb{S}$ to formulate as,

$$\mathbb{C} = \{r_1, r_2, ..., r_N\}, \quad \mathbb{S} = \{(U_{r_1}, V_{r_1}), (U_{r_2}, V_{r_2}), ..., (U_{r_N}, V_{r_N})\} \tag{1}$$

During each forward pass for supernet training, when we select the rank level to be r_i, we pick the corresponding building block (U_{r_i}, V_{r_i}) as low-rank weights, and calculated with:

$$\text{LRLinear}(x, r_i) = U_{r_i} V_{r_i}^T x + b, \tag{2}$$

where x is the input for the current linear components and b is the bias term originally encoded in this components.

Next, we can formulate the search space $\mathcal{A}$ for our low-rank one-shot NAS as

$$\mathcal{A} = \{(r^{(1)}, ..., r^{(i)}, ..., r^{(l)}) | r^{(i)} \in \mathbb{C}^{(i)}, 1 \le i \le l\}, \tag{3}$$

where $\mathbb{C}^{(i)}$ includes all possible rank choices of the i-th choice block, $r^{(i)}$ is the rank choice of the i-th choice block, and l is the number of choice blocks.

3.2 BUILDING LOW-RANK SUPERNETS

Weight Inheritance Low-Rank Linear Layer Now that we have the search space of low-rank decompositions U_{r_i} V_{r_i} designated with rank choice r_i, we need to design a suitable implementation for the low-rank choice block. A vanilla version would be following the weight sharing strategy (Bender et al., 2018) by starting from a supernet and selecting from its subsets. For each building block (U_{r_i}, V_{r_i}) in the supernet, we factorize the weights of corresponding linear layers from the uncompressed model into two low-rank matrices as the pre-trained weights, and concatenated them into a huge choice block. For each components, the weights of building blocks within the same choice block are mutually exclusive, demonstrated as:

$$U_{r_i} \cap U_{r_j} = \emptyset, \quad V_{r_i} \cap V_{r_j} = \emptyset, \quad \forall r_i, r_j \in \mathbb{C},\, r_i \neq r_j \tag{4}$$

This vanilla version of concatenating substructure has two main defect in practical use. At one hand, the concatenation in all makes the super-structure too bloated to be trained under limited resource budget. On the other hand, for each training step, only the weights in the chosen building blocks will be activated, and gradients will only back-propagate to a single building block. This makes the training procedure very inefficient and suboptimal without sharing the gradient, leading to poor convergence overall Chen et al. (2021b) has also observed the same downsides of this strategy.

Inspired by Single-Path NAS (Stamoulis et al., 2019), we incorporate the separated building blocks into a single superblock, where the low-ranks weights of each building block are actually the subset of the largest ones. Here, we propose weight inheritance to enable weight sharing between building blocks during training, addressing both the problem of the bloated super-structure and sub-optimal gradients. The structure is shown in Fig. 1-a. Concretely, within each choice block, the building block with the largest rank choice (U_{r_N}, V_{r_N}) is set to the size of the factorized weights of the corresponding linear layer in the uncompressed model and initialized with the SVD. Details shown in A.2. Under classical calculation of a SVD, eigenvalues are ranked from the largest to the smallest to the diagonal. Thus, it is intuitive to keep the top eigenvectors even under the lowest rank choice. Following this instruction, we design the building blocks under a bottom-to-top regime: for the choice block of rank level r_i, we select the top r_i columns from $U_{r_{i+1}}$, $V_{r_{i+1}}$, which can be formulated as:

$$U_{r_i} = U_{r_{i+1}}[:, : r_i], \quad V_{r_i} = V_{r_{i+1}}[:, : r_i] \tag{5}$$

It is easy to see that each sub-structure is also the sub-matrix of the all the structure with larger rank level, including the super-matrix:

$$U_{r_i} \subseteq U_{r_k}, \quad V_{r_i} \subseteq V_{r_k}, \quad \forall k \in \{i+1, i+2, ..., N\} \quad \forall r_i \in \mathbb{C} \tag{6}$$

Thus, in one forward pass, the values of U_{r_i} and V_{r_i} are inherited from the first-r_i column vectors of U_{r_N} and V_{r_N} respectively; in a backward pass, the gradients of U_{r_i} and V_{r_i} are updated back to the corresponding sub-matrices in U_{r_N} and V_{r_N}. Since the weights of each building block are shared, the gradient information can be leveraged among the groups. Once we update the weights of the smaller building blocks, the larger ones will be updated synchronously, and vice versa. Under the weight inheritance strategy, all building blocks can be trained simultaneously, resulting in better convergence on the supernet training overall (refer to Sec 4.4 for more detailed analysis.).

Finally, we can replace the linear projection modules in the MHSA and MLP modules with the low-rank linear layer to construct the supernet for low-rank optimization. (Please refer to Appendix A.3 for more details)

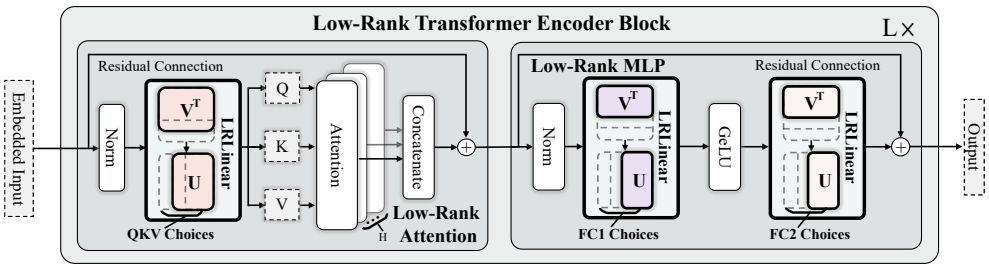

Figure 2: Modified encoder block of the supernet.

3.3 TRAINING LOW-RANK SUPERNETS

After performing Stage I, we encode the low-rank search space into the supernet, as shown in Fig. (2). In stage II, we change the original training objective function from SPOS, formulated as:

$$W_{\mathcal{A}} = \underset{W}{\text{argmin}} \mathbb{E}_{a \sim \Gamma(\mathcal{A})}[\mathcal{L}_{\text{soft}}(\mathcal{N}(a, W(a))], \tag{7}$$

where $W_{\mathcal{A}}$ is the weights of the supernet, $a \in \mathcal{A}$ is a rank configuration, $W(a)$ is weights of a, $\mathcal{N}(a, W(a))$ denotes the corresponding subnetwork, and $\mathcal{L}_{\text{soft}}$ is the cross-entropy loss with teacher generated soft labels on training set. In each training step, we select a subnetwork from the prior distribution $\Gamma(\mathcal{A})$ with non-uniform sampling, train it with the teacher-generated soft label, and update weights back in the supernet, as shown in Fig. 1-b. Once the supernet has converged, we can acquire subnetworks by inheriting the weights from the supernet directly.

Non-Uniform Sampling Unlike previous works (Guo et al., 2020; Chen et al., 2021b) that train the supernet from scratch, our supernet is initialized with the weight of uncompressed pre-trained model, and the subnetworks are the low-rank approximated networks for various rank choices. Thus, during the training process, it is no longer a good choice for subnetworks to be sampled under an uniform distribution. We want to design a new sampling distribution that mainly under the guidance of two observations:

- Architecture search algorithm are in general large and hard to train. For a pre-selected numerous search space, covering all the subnetworks and make every sub structure well-trained is a intractable task. To enable the overall structure to reach optimal performance and decent covergence, we need to reasonably distribute the training resources to different subnetworks under a non-uniform distribution.
- According to the design of weight inheritance, we know that a small submatrix at rank level i are the subsets of all the matrix with rank level great than i (Eq. 6). Then, the lower the rank level of a submatrix is, the higher chance for the subnetworks to be shared in the search space. Hence, it is much more essential to allocate more training resources (sampling probability) to the smaller structures.

Following the observations mentioned above, we propose a simple but very effective non-uniform path sampling strategy, which gives the smaller rank choices of each low-rank linear layer a higher probability of being sampled. Mathematically, the rank choice for each low-rank linear layer can be expressed by a random variable X. The PMF of X is formulated as:

$$p_X(r) = P(X = r) = \frac{\frac{1}{r}}{\sum_{r' \in \mathbb{C}} \frac{1}{r'}}, \quad r \in \mathbb{C} \tag{8}$$

Remember that $\mathbb{C}$ denotes the rank choice set of a specific low-rank linear layer. The prior distribution of selecting a sequence of rank choice would be $\Gamma(\mathcal{A}) = P(X_1 = r_{k_1}, ..., X_i = r_{k_i}, ..., X_l = r_{k_l})$, where X_i, r_{k_i} denotes the random variable of i-th choice block and its correlated rank choice, individually.

Soft Label Distillation To ensure all compressed subnetworks inside the supernet are well-trained, we integrate knowledge distillation into our framework. Here, we conduct an empirical study on the influence of knowledge distillations (see appendix A.5 for more details) and observe that knowledge distillation makes information recovery easier; also, we find that using only the teacher-generated soft labels can yield better results than using it along with hard labels, which was widely used in prior works (Yuan et al., 2020; Touvron et al., 2021; Yu et al., 2022b). To generalize our framework, we select the uncompressed model itself for the supervising task.

3.4 SEARCHING FOR OPTIMAL RANK CONFIGURATION

In stage III, we aim to maximize the model performance under FLOPs constraints:

$$a^* = \underset{a \in \mathcal{A}}{\text{argmax}} \text{ACC}_{\text{val}}(\mathcal{N}(a, W_{\mathcal{A}}(a))) \quad \text{s.t.} \quad \mathcal{F}_{lower} \leq \mathcal{F}(a) \leq \mathcal{F}_{upper}, \tag{9}$$

where ACC_{val} is the accuracy on validation set, $\mathcal{F}(a)$ denotes the FLOPs of the subnetwork with the rank configuration a, $\mathcal{F}_{lower}$ and $\mathcal{F}_{upper}$ are the lower bound and upper bound of FLOPs constraints, respectively.

Here, we adopt an evolutionary algorithm to search for the proper rank choices, with the workflow shown in Fig. 1-c. We take every rank choice r as genes and every rank configuration a as chromosomes. In the beginning, we randomly sample and pick N_p rank configurations under the FLOPs constraint as the initial population. During each epoch, we select the top-k rank configurations from the population as parents and execute mutation and crossover operations to generate the new rank configurations. Next, we add the new rank configurations that satisfy the FLOPs constraint, to the population of the next generation. Finally, we apply random sampling to produce more rank configurations by ensuring the population count is N_p. After N_e epochs, the evolutionary algorithm is finished, and we can acquire the proper rank configuration.

4 EXPERIMENTS

We train and test the Trimsformer on ImageNet-1k (Deng et al., 2009) dataset with several representative ViT models, including DeiT (Touvron et al., 2021), Swin Transformer (Liu et al., 2021a), as our pre-trained backbone. For the details of hyperparameter used in training and supernet configuration please refer to appendix A.4.

Table 1: Comparison of Trimsformer with different ViT compression methods on ImageNet-1k dataset. "Top-1 Acc diff" denotes accuracy improvement over the baseline. "Ours" denotes the model obtained on Trimsformer

Base-model	Method	FLOPs	FLOPs saving	Params	Params saving	Top-1 Acc	Top-1 Acc diff
DeiT-S	Baseline	4.7G	-	22.1M	-	79.8%	-
	DynamicViT	3.4G	28%	23.1M	-	79.6%	-0.2%
	SPViT	3.3G	30%	16.4M	26%	78.3%	-1.5%
	WDPruning	3.1G	32%	15.0M	32%	78.6%	-1.2%
	S^2ViTE	3.1G	32%	14.6M	34%	79.2%	-0.6%
	UVC	2.7G	42%	-	-	79.4%	-0.4%
	Ours	3.1G	32%	14.5M	34%	79.4%	-0.4%
DeiT-B	Baseline	17.6G	-	86.4M	-	81.8%	-
	S^2ViTE	11.8G	33%	56.8M	35%	82.2%	+0.4%
	SPViT	11.7G	33%	62.3M	28%	81.6%	-0.2%
	DynamicViT	11.2G	36%	87.2M	-	81.3%	-0.5%
	WDPruning	11.0G	37%	60.6M	30%	81.1%	-0.7%
	UVC	8.0G	55%	-	-	80.6%	-1.2%
	Ours	11.1G	36%	53.6M	38%	82.2%	+0.4%
	Ours	8.0G	55%	37.9M	56%	81.5%	-0.3%
Swin-S	Baseline	8.7G	-	49.6M	-	83.2%	-
	DynamicViT	6.9G	20%	50.8M	-	83.2%	0.0%
	WDPruning	6.8G	22%	37.4M	26%	82.4%	-0.8%
	SPViT	6.1G	30%	39.2M	30%	82.4%	-0.8%
	Ours	6.6G	25%	37.3M	26%	83.2%	0.0%
	Ours	6.1G	30%	34.8M	30%	82.8%	-0.4%
Swin-B	Baseline	15.4G	-	87.8M	-	83.5%	-
	DynamicViT	12.1G	21%	88.8M	-	83.4%	-0.1%
	SPViT	11.4G	26%	68.0M	24%	83.2%	-0.3%
	Ours	11.4G	26%	63.5M	28%	83.7%	+0.2%

4.1 RESULTS – COMPARISON WITH DIFFERENT COMPRESSION METHODS

We compare our results with state-of-the-art ViT pruning methods, ranging from input sequence reduction (DynamicViT (Rao et al., 2021)), weight pruning (WDPruning (Yu et al., 2022a), S^2ViTE (Chen et al., 2021d)), and multi-dimension pruning (UVC (Yu et al., 2022b).

The results are shown in Tab. 1. Compared to weight pruning methodologies (WDPruning), we achieve around 1% accuracy improvement under the same or smaller level of computation reduc-

tion and model size on every backbone. Secondly, when considering the sequence reduction-based methods (DynamicViT), we demonstrate comparable FLOPs savings on all different optimized targets and reduce the parameters significantly. For DeiT-B/Swin-B, we can enjoy 20%/6% additional FLOPs reduction and an extra 60%/30% model size saving when compared to DynamicViT. Lastly, compared to multi-dimensional compression methods, we achieve better results on DeiT-Base with 1% improvement on performance under the same FLOPs level. Overall, we can observe that Trimsformer shows competitive performance with SOTA methods on small models like DeiT-S and Swin-S while outperforms all the compared methods on large models like DeiT-B and Swin-B.

4.2 Results – Searching for the Optimal Architecture

The effectiveness of Trimsformer on ViT compression have already been demonstrated above, now we want to unleash the potential of Trimsformer on constructing an optimal architecture. To push the Pareto frontier further up, we want to search for an optimal architecture that is of both lighter computation and higher performance than current backbones. To further improve the accuracy while keep the FLOPs low, we replace the teacher model with Swin Transformer (Liu et al., 2021a) for supervising. In Tab. 2, it is demonstrated that Trimsformer can push the current SOTA transformer backbones to around 1% performance improvement while saving on average 30% of the computational cost for smaller model. On larger models like Swin-B and DeiT-B, the margin is pushed further with at most 57% FLOPs saving while improving 1.1% top-1 accuracy for DeiT-B. We further describe a Pareto frontier in Fig. 3. Under the structure design of Trimsformer, vanilla ViT architecture(ViT, DeiT) can achieve competitive results with the Swin Transformer family even without the specialized shifted window strategy. When the Trimsformer is applied on Swin Transformer backbone, we develop a superior set of models that achieves the best trade-off between FLOPs and accuracy. Trim-Swin-B surpass the margins of S3 (Chen et al., 2021c) and MiniViT (Zhang et al., 2022), where both of them developed new architecture based on Swin backbone.

Table 2: Result of optimal Trimsformer architecture on ImageNet-1k dataset.

Models	Top-1 Acc	Top-1 Acc diff	FLOPs	FLOPs saving	Params	Params saving
DeiT-S	79.8%	-	4.7G	-	22M	-
Trim-DeiT-S	81.0%	+1.2%	4.1G	14%	19M	14%
Trim-DeiT-S	80.5%	+0.7%	3.3G	30%	15M	32%
Swin-T	81.3%	-	4.5G	-	28M	-
Trim-Swin-T	82.2%	+0.9%	3.6G	20%	23M	17%
Trim-Swin-T	81.5%	+0.2%	2.9G	36%	19M	32%
Swin-S	83.0%	-	8.7G	-	50M	-
Trim-Swin-S	83.8%	+0.8%	6.5G	25%	38M	25%
Trim-Swin-S	83.2%	+ 0.2%	5.1G	41%	29M	42%
DeiT-B	81.8%	-	17.6G	-	87M	-
Trim-DeiT-B	83.7%	+1.9%	10.5G	40%	50M	42%
Trim-DeiT-B	82.9%	+0.9%	7.5G	57%	36M	59%
Swin-B	83.3%	-	15.4G	-	88M	-
Trim-Swin-B	84.4%	+1.1%	9.5G	38%	54M	39%
Trim-Swin-B	83.6%	+0.8%	8.3G	46%	48M	46%

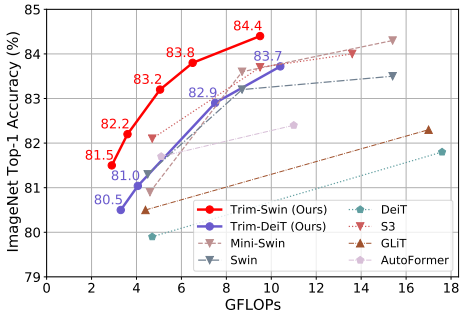

Figure 3: Pareto frontier comparison curve for Trimsformer with SOTA Vision Transformers.

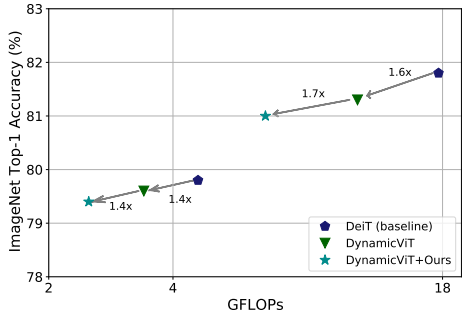

Figure 4: Results of integrating Trimsformer with DynamicViT.

4.3 GENERALITY AND ORTHOGONALITY

Results on the Compact Hybrid Transformer In the previous section, we have demonstrated the generality of Trimsformer on Transformer-only variants DeiT and Swin Transformer. Here we further show that our approach is agnostic to model architectures even on a hybrid structure (*i.e.*, CNN + Transformer). The results of trimming a TinyViT-22M with 224 x 224 input is shown in Tab. 3. The computational cost can be reduced by 25% without sacrificing performance on TinyViT, affirming the generality of our proposed Trimsformer.

Integrating with Other Compression Methods To show the orthogonality of our approach, we integrate the proposed Trimsformer with token reduction based compression method DynamicViT to achieve further compression ratio. Here, we use the uncompressed model itself for supervising. From Tab. 3, Trimsformer achieves 21%/26% more FLOPs reduction while only sacrificing 0.3% more performance degradation on DeiT-S/B. Fig. 4 provides an clearer view for the demonstration of orthogonality. On DeiT-B, when integrating Trimsformer with DynamicViT, the reduction on FLOPS is further improved by 1.7x over standalone DynamicViT. Moreover, on the small-scale model DeiT-S, we can also earn 1.4x more FLOPs improvement and a compelling 50% FLOPs savings in total. Thus, Trimsformer provides a new perspective that is orthogonal to the previous compression methods. By applying multiple compression techniques, ViT can be compressed to an appealing ratio with less than 50% of the original FLOPs and less than 1% drop on performance.

Table 3: Results on TinyViT and integration with DynamicViT. DynamicViT is denoted as DyViT. The model with Trim- prefix are the model generated by Trimsformer.

Model	GFLOPs	Top-1 Acc
TinyViT-21M	4.3G	83.1%
Trim-TinyViT-21M	3.2G (-25%)	83.1% (-0%)
DeiT-S	4.7G	79.8%
DyViT-DeiT-S	3.4G (-28%)	79.6% (-0.3%)
Trim-DyViT-DeiT-S	2.3G (-49%)	79.3% (-0.6%)
DeiT-B	17.6G	81.8%
DyViT-DeiT-B	11.2G (-36%)	81.3% (-0.5%)
Trim-DyViT-DeiT-B	6.70G (-62%)	81.0% (-0.8%)

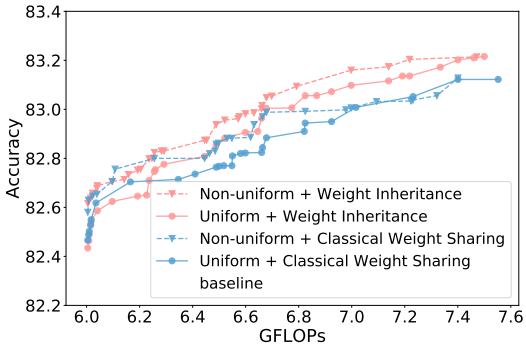

Figure 5: Pareto frontier of Trimsformer with Swin-S backbone using different configurations.

4.4 ABLATION STUDY

Effect of Non-Uniform Sampling By considering the initial performance of the subnetworks, a non-uniform sampling strategy can make a better allocation of the total supernet training resource, allocating more training steps to those subnetworks with more pre-trained weights pruned. As shown in Fig. 5, the smaller subnetworks can achieve higher accuracy than the uniform sampling, which is preferable for pruning purposes.

Effect of Weight Inheritance From Sec. 3, we know that weight inheritance make the supernet slim and converges better. We demonstrate its effectiveness in Fig. 5. It can be observed that weight inheritance with either uniform or non-uniform sampling converges to the baseline at around 7 to 7.5 GFLOPs, while classical weight sharing cannot.

Finally, when the two proposed strategies are conducted simultaneously, the resulting configurations sit on the Pareto frontier. Reader can find more ablation studies in appendix A.5.

5 CONCLUSION

This paper proposes Trimsformer, an end-to-end framework that automatically searches for optimal low-rank configuration. Extensive experiments show strong performance and good generality. Extending our framework for multi-dimensional compression can be an exciting direction in the future.

REFERENCES

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Understanding and simplifying one-shot architecture search. In Jennifer G. Dy and Andreas Krause (eds.), *ICML*, 2018.

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

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

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.

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

Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *ICCV*, pp. 12250–12260, 2021b.

Minghao Chen, Kan Wu, Bolin Ni, Houwen Peng, Bei Liu, Jianlong Fu, Hongyang Chao, and Haibin Ling. Searching the search space of vision transformer. In *NeurIPS*, pp. 8714–8726, 2021c.

Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. In *NeurIPS*, pp. 19974–19988, 2021d.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 2019.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *ICLR*, 2020.

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

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. DynaBERT: Dynamic BERT with adaptive width and depth. In *NeurIPS*, 2020.

Zejiang Hou and Sun-Yuan Kung. Multi-dimensional vision transformer compression via dependency guided gaussian process search. In *CVPR*, pp. 3668–3677, 2022.

Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán. Low-rank compression of neural nets: Learning the rank of each layer. In *CVPR*, 2020.

Ding Jia, Kai Han, Yunhe Wang, Yehui Tang, Jianyuan Guo, Chao Zhang, and Dacheng Tao. Efficient vision transformers via fine-grained manifold distillation. *arXiv preprint arXiv:2107.01378*, 2021.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In Yoshua Bengio and Yann LeCun (eds.), *ICLR*, 2016.

Zhenglun Kong, Peiyan Dong, Xiaolong Ma, Xin Meng, Wei Niu, Mengshu Sun, Bin Ren, Minghai Qin, Hao Tang, and Yanzhi Wang. Spvit: Enabling faster vision transformers via soft token pruning. *CoRR*, abs/2112.13890, 2021.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In Yoshua Bengio and Yann LeCun (eds.), *ICLR*, 2015.

Dongsoo Lee, Se Jung Kwon, Byeongwook Kim, and Gu-Yeon Wei. Learning low-rank approximation for cnns. *CoRR*, 2019.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, pp. 9992–10002, 2021a.

Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. *Advances in Neural Information Processing Systems*, 34:28092–28103, 2021b.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, pp. 2755–2763, 2017.

Sachin Mehta and Mohammad Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *ICLR*, 2022.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient vision transformers with dynamic token sparsification. In *NeurIPS*, pp. 13937–13949, 2021.

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path NAS: designing hardware-efficient convnets in less than 4 hours. In Ulf Brefeld, Élisa Fromont, Andreas Hotho, Arno J. Knobbe, Marloes H. Maathuis, and Céline Robardet (eds.), *ECML PKDD*, 2019.

Cheng Tai, Tong Xiao, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization. In Yoshua Bengio and Yann LeCun (eds.), *ICLR*, 2016.

Yehui Tang, Kai Han, Yunhe Wang, Chang Xu, Jianyuan Guo, Chao Xu, and Dacheng Tao. Patch slimming for efficient vision transformers. *CoRR*, abs/2106.02852, 2021.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, pp. 10347–10357, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pp. 5998–6008, 2017.

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*, pp. 7675–7688, 2020.

Yixuan Wei, Han Hu, Zhenda Xie, Zheng Zhang, Yue Cao, Jianmin Bao, Dong Chen, and Baining Guo. Contrastive learning rivals masked image modeling in fine-tuning via feature distillation. *CoRR*, abs/2205.14141, 2022.

Genta Indra Winata, Samuel Cahyawijaya, Zhaojiang Lin, Zihan Liu, and Pascale Fung. Lightweight and efficient end-to-end speech recognition using low-rank transformer. In *ICASSP*, pp. 6144–6148, 2020.

Kan Wu, Jinnian Zhang, Houwen Peng, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Tinyvit: Fast pretraining distillation for small vision transformers. *CoRR*, abs/2207.10666, 2022.

Zhuofan Xia, Xuran Pan, Shiji Song, Li Erran Li, and Gao Huang. Vision transformer with deformable attention. *CoRR*, abs/2201.00520, 2022.

Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai XiongA vit: Adaptive tokens for efficient vision transforme. TRP: trained rank pruning for efficient deep neural networks. In Christian Bessiere (ed.), *IJCAI*, 2020.

Hongxu Yin, Arash Vahdat, Jose M Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-vit: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Fang Yu, Kun Huang, Meng Wang, Yuan Cheng, Wei Chu, and Li Cui. Width & depth pruning for vision transformers. In *AAAI*, pp. 3143–3151, 2022a.

Shixing Yu, Tianlong Chen, Jiayi Shen, Huan Yuan, Jianchao Tan, Sen Yang, Ji Liu, and Zhangyang Wang. Unified visual transformer compression. In *ICLR*, 2022b.

Li Yuan, Francis E. H. Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. In *CVPR*, pp. 3902–3910, 2020.

Zhihang Yuan, Chenhao Xue, Yiqi Chen, Qiang Wu, and Guangyu Sun. Ptq4vit: Post-training quantization framework for vision transformers. *arXiv preprint arXiv:2111.12293*, 2021.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. *CoRR*, 2021.

Jinnian Zhang, Houwen Peng, Kan Wu, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Minivit: Compressing vision transformers with weight multiplexing. *CoRR*, abs/2204.07154, 2022.

Mingjian Zhu, Kai Han, Yehui Tang, and Yunhe Wang. Visual transformer pruning. *arXiv preprint arXiv:2104.08500*, 2021.

A APPENDIX

A.1 ANALYSIS OF COMPUTATIONAL COST

Table 4: FLOPs analysis. Linear projections account for a high percentage of the total computation cost in DeiT-B and Swin-B.

Model	#	MHSA linear projections	MLP linear projections	Others	Total
DeiT-B	FLOPs	5.58G (32%)	11.15G (64%)	0.52G (4%)	17.56G
	Params	28.31M (32%)	56.62M (66%)	1.38M (2%)	86.31M
Swin-B	FLOPs	4.93G (32%)	9.87G (63%)	0.67G (5%)	15.47G
	Params	27.92M (32%)	55.83M (64%)	3.81M (4%)	87.56M

A.2 RANK-K LOW-RANK APPROXIMATION

Firstly, we decompose W into three matrices $\hat{U} \in \mathbb{R}^{m\times r}$, $\hat{\Sigma} \in \mathbb{R}^{r\times r}$, and $\hat{V} \in \mathbb{R}^{n\times r}$ by singular value decomposition, where $r = min(m, n)$, $\hat{\Sigma} = diag(\sigma_1, \sigma_2, ..., \sigma_r)$, and $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r \geq 0$

$$[\hat{U}, \hat{\Sigma}, \hat{V}] = \text{SVD}(W) \tag{10}$$

Secondly, we fuse $\hat{\Sigma}$ into $\hat{U}$ or $\hat{V}$ to reduce the number of parameters by the following equation.

$$U = \begin{cases} \hat{U}\hat{\Sigma} & , n > m \\ \hat{U} & , m \geq n \end{cases}, V^T = \begin{cases} \hat{V}^T & , n > m \\ \hat{\Sigma}\hat{V}^T & , m \geq n \end{cases} \tag{11}$$

Lastly, we pick the first k columns from U and V to construct the U_k and V_k.

$$U_k = U[:, : k], \quad V_k = V[:, : k] \tag{12}$$

A.3 DETAILS OF BUILDING SUPERNETS

In the following two paragraphs, we will show the way to incorporate the weight inheritance low-rank linear layers into Attention and MLP, which are two primary components of ViT, to build the low-rank supernet.

Low-Rank Linear Layer in Attention A typical attention module in transformers is mainly composed of three learnable weight matrices which perform the linear transformation to generate the query ($W^Q \in \mathbb{R}^{n\times d}$), key ($W^K \in \mathbb{R}^{n\times d}$), and value ($W^V \in \mathbb{R}^{n\times d}$) embeddings as well as the self-attention module that aggregate the global correlations. In practice, the three learnable weight matrices can be combined into a single large weight matrix $W^{QKV} \in \mathbb{R}^{n\times 3d}$ since the inputs are identical, where n is the number of input patches, and d is the embedding dimension of input patches. We can replace the weight matrix W^{QKV} with a weight inheritance low-rank linear layer LRLinear^{QKV} to reduce the operations and parameters of the attention module.

The number of activated parameters in LRLinear$^{\text{QKV}}$ is $4nr$, where r is the given rank choice of W^{QKV}. If we choose a sufficient small $r < \frac{3}{4}n$, the number of parameters and FLOPs are smaller compared to the uncompressed one.

$$[Q, K, V] = \text{LRLinear}^{\text{QKV}}(x, r) \tag{13}$$

Notably, this modification can be generally applied to various type of attention designs that required linear embeddings for query, key and value, such as deformable ttention (Xia et al., 2022) or shifted-window based self-attention (Liu et al., 2021a).

Low-Rank Linear Layer in Multi-Layer Perceptron A typical MLP layer in transformers contains two linear layers and the GELU activation. We can replace linear layers with weight inheritance low-rank linear layers to reduce the computational cost.

$$\text{MLP}(x, r_1, r_2) = x + \text{LRLinear}^{\text{FC2}}(\text{GELU}(\text{LRLinear}^{\text{FC1}}(x, r_1), r_2) \tag{14}$$

The number of activated parameters in a weight inheritance low-rank linear layer is $5nr$, when MLP ratio is equal to 4, where r is the given rank choice of the weight matrix. If we choose a sufficient small $r < \frac{4}{5}n$, the number of parameters and FLOPs are smaller compared to the uncompressed one.

Table 5: Configuration for supernet. $\mathcal{R}$ is the number of the full rank for each layer.

Backbone	KD-type	Rank choices	Params range
Swin-T	Uncompressed one	$[\frac{9}{16}\mathcal{R}, \frac{10}{16}\mathcal{R}, \frac{11}{16}\mathcal{R}, \mathcal{R}]$	22M∼28M
	Swin-L	$[\frac{7}{16}\mathcal{R}, \frac{8}{16}\mathcal{R}, \frac{9}{16}\mathcal{R}, \mathcal{R}]$	18M∼28M
Swin-S	Uncompressed one	$[\frac{8}{16}\mathcal{R}, \frac{9}{16}\mathcal{R}, \frac{10}{16}\mathcal{R}, \mathcal{R}]$	34M∼50M
	Swin-L	$[\frac{6}{16}\mathcal{R}, \frac{7}{16}\mathcal{R}, \frac{8}{16}\mathcal{R}, \mathcal{R}]$	27M∼50M
Swin-B	Uncompressed one	$[\frac{8}{16}\mathcal{R}, \frac{9}{16}\mathcal{R}, \frac{10}{16}\mathcal{R}, \mathcal{R}]$	60M∼88M
	Swin-L	$[\frac{6}{16}\mathcal{R}, \frac{7}{16}\mathcal{R}, \frac{8}{16}\mathcal{R}, \mathcal{R}]$	48M∼88M
DeiT-S	Uncompressed one	$[\frac{7}{16}\mathcal{R}, \frac{8}{16}\mathcal{R}, \frac{9}{16}\mathcal{R}, \frac{10}{16}\mathcal{R}, \mathcal{R}]$	13M∼22M
	Swin-L	$[\frac{5}{12}\mathcal{R}, \frac{6}{12}\mathcal{R}, \frac{7}{12}\mathcal{R}, \frac{8}{12}\mathcal{R}, \mathcal{R}]$	13M∼22M
DeiT-B	Uncompressed one	$[\frac{13}{48}\mathcal{R}, \frac{17}{48}\mathcal{R}, \frac{21}{48}\mathcal{R}, \frac{25}{48}\mathcal{R}, \mathcal{R}]$	36M∼86M
	Swin-L	$[\frac{11}{48}\mathcal{R}, \frac{15}{48}\mathcal{R}, \frac{19}{48}\mathcal{R}, \frac{23}{48}\mathcal{R}, \frac{27}{48}\mathcal{R}]$	31M∼64M

Table 6: Hyper-parameter settings for training

Model family	Epochs	Learning rate	Weight decay	Drop path	CutMix	Mixup	Repeated augmentation
Swin	150	$1 * 10^{-5}$	0.01	0.1	✗	✗	✓
DeiT	300	$7.5 * 10^{-6}$	0.01	0.0	✗	✗	✓

A.4 Implementation Details

The search space for each supernet with different backbone and distillation types are defined in Tab. 5. Here, we set rank choices for QKV, FC1, and FC2 to be identical for a simple implementation; however, we can set them to be different to support more complicated configurations. For training, we adopt a similar recipe to the backbone, and Tab. 5,6 provide the details for the hyperparameter setting. Here, we set a relatively small learning rate for fine-tuning since the weights are derived from the pre-trained model. For the implementation of the soft label distillation training we adopt the fast-distillation framework from TinyViT (Wu et al., 2022).

To implement the evolutionary algorithm, we set the number of epochs N_e to be 20 and the population size N_p to be 50. Every generation, the top-10 configurations are selected to perform mutation with probability P_m set to be 0.2. The number of chromosomes generated by mutation (N_m) and crossover (N_c) is set to be 25.

A.5 More Ablation Study

Effect of Neural Architecture Search on Low-Rank Approximation We re-implement the LRT as our comparison baseline to investigate the effect of our NAS-based low-rank approximation. We

Table 7: Ablation study. The top-1 accuracy is averaged over 400 random sampled sub-networks from the Trimsformer supernet with Swin-S backbone on ImageNet-1k. WS: Weight sharing. SS: Sampling strategy. KD: Knowledge distillation. Hard: Hard label. Soft: Soft label. US: Uniform Sampling. NUS: Non-Uniform Sampling. CWS: Classical Weight Sharing. WI: Weight Inheritance. Note that CWS + US reduces to the vanilla one-shot NAS.

#	WS		SS		KD		Supernet parameters	Average Top-1 Acc
	CWS	WI	US	NUS	Hard	Soft		
1	✓		✓			✓	142M	82.76% ±0.16%
2	✓			✓		✓	142M	82.82% ±0.13%
3		✓	✓			✓	61M	82.84% ±0.19%
4		✓		✓	✓	✓	61M	82.73% ±0.18%
5		✓		✓		✓	61M	82.91% ±0.16%

Table 8: Comparison of methodologies for rank decision. We report the top-1 accuracy of the low-rank approximated Swin-S model on ImageNet. The GPU hours are measured on V100 GPU. N denotes the number of deployment scenarios.

Method	FLOPs	ACC	Configuration design type	Training cost (GPU hours)	Search cost (GPU hours)	Total
LRT	6.6G	83.1%	Manual	$208N$	0	$208N$
Ours	6.1G	82.8%	Auto	208	$24N$	208+$24N$
Ours	6.6G	83.2%				
Ours	7.0G	83.4%				

train both networks with soft label-only knowledge distillation for fair comparison under the same target GFLOPs. We summarize the comparison in Tab. 8.

As shown in Tab. 8, first, we can see that the training cost of LRT is proportional to the number of deployment scenarios, while Trimsformer remains constant and can generate accuracy-FLOPs trade-offs by training only once. Second, due to the lack of a proper searching method, LRT is coarse-grained and requires the user to set the rank manually, while Trimsformer can generate the configurations automatically; besides, the configurations can be non-uniform, providing a fine-grained approximation with better performance.

Effect of Soft Label Distillation From the ablation study presented in Table 7, we observe that training with only the soft label can yield a supernet with better performance; using an extra ground truth label leads to a 0.2% performance degradation on average for the subnets of Swin-S. This phenomenon indicates that ground truth labels may not be a proper guideline for the compressed small networks because they may be too assertive and, thus, suppress the learning effect during fine-tuning.