MiniViT: Compressing Vision Transformers with Weight Multiplexing

Jinnian Zhang[1,*], Houwen Peng[1,*,†], Kan Wu[1,*], Mengchen Liu[2], Bin Xiao[2], Jianlong Fu[1], Lu Yuan[2]
[1] Microsoft Research, [2] Microsoft Cloud+AI
{v-jinnizhang, houwen.peng, v-kanwu, mengcliu, bin.xiao, jianf, luyuan}@microsoft.com

Abstract

Vision Transformer (ViT) models have recently drawn much attention in computer vision due to their high model capability. However, ViT models suffer from huge number of parameters, restricting their applicability on devices with limited memory. To alleviate this problem, we propose MiniViT, a new compression framework, which achieves parameter reduction in vision transformers while retaining the same performance. The central idea of MiniViT is to multiplex the weights of consecutive transformer blocks. More specifically, we make the weights shared across layers, while imposing a transformation on the weights to increase diversity. Weight distillation over self-attention is also applied to transfer knowledge from large-scale ViT models to weight-multiplexed compact models. Comprehensive experiments demonstrate the efficacy of MiniViT, showing that it can reduce the size of the pre-trained Swin-B transformer by 48%, while achieving an increase of 1.0% in Top-1 accuracy on ImageNet. Moreover, using a single-layer of parameters, MiniViT is able to compress DeiT-B by 9.7 times from 86M to 9M parameters, without seriously compromising the performance. Finally, we verify the transferability of MiniViT by reporting its performance on downstream benchmarks. Code and models are available at here.

1. Introduction

"Only Mini Can Do It."

— BMW Mini Cooper

Large-scale pre-trained vision transformers, such as ViT [18], CvT [58], and Swin [36], have recently drawn a great deal of attention due to their high model capabilities and superior performance on downstream tasks. However, they generally involve giant model sizes and large amounts of pre-training data. For example, ViT uses 300 million images to train a huge model with 632 million parameters, achieving state-of-the-art performance on image classification [18]. Meanwhile, the Swin transformer uses 200-300 million parameters, and is pre-trained on ImageNet-22K [16], to attain promising results on downstream detection and segmentation tasks [36].

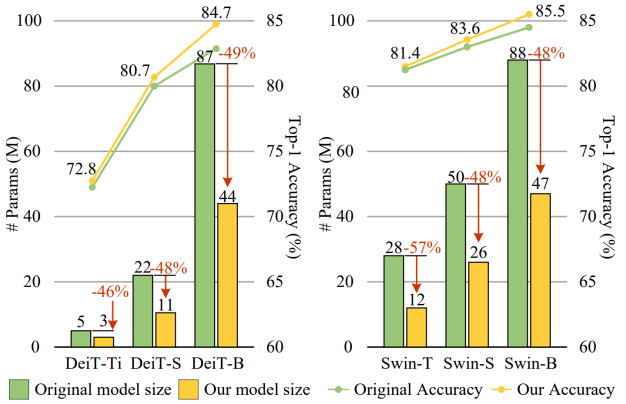


Figure 1. Comparisons between MiniViTs and popular vision transformers, such as DeiT [52] and Swin Transformers [36]

Hundreds of millions of parameters consume considerable storage and memory, making these models unsuitable for applications involving limited computational resources, such as edge and IoT devices, or in which real-time predictions are needed. Recent studies reveal that the large-scale pre-trained models are over-parametrized [30] . Therefore, it is necessary and feasible to eliminate redundant parameters and the computational overhead of these pre-trained models without compromising their performance.

Weight sharing is a simple, but effective, technique to reduce model sizes. The original idea of weight sharing in neural networks was proposed in the 1990s by LeCun and Hinton [33, 40], and recently reinvented for transformer model compression in natural language processing (NLP) [32]. The most representative work, ALBERT [32], introduces a cross-layer parameter sharing method to prevent the number of parameters from growing with network depth. Such technique can significantly reduce the model size without seriously hurting performance, thus improving parameter efficiency. However, the efficacy of weight sharing in vision transformer compression is not well explored.

To examine this, we perform the cross-layer weight sharing [32] on DeiT-S [52] and Swin-B [52] transformers. Unexpectedly, this straightforward usage of weight sharing brings two severe issues: (1) *Training instability.* We ob-

*Equal contributions. Work done when Jinnian and Kan were interns of Microsoft. †Corresponding author.

served that weight sharing across transformer layers makes the training become unstable, and even causes training collapse as the number of shared layers increases, as visualized in Fig. 4. (2) *Performance degradation.* The performance of weight-shared vision transformers drops significantly compared to the original models. For example, it leads to a 5.6% degradation in accuracy for Swin-S, although weight sharing can reduce the number of model parameters by fourfold.

To investigate the underlying reasons for these observations, we analyze the ℓ_2-norm of gradients during training and the similarities between intermediate feature representations from the model before and after weight sharing (cf. Sec. 4.2). We found that strictly identical weights across different layers is the main cause of the issues. In particular, the layer normalization [5] in different transformer blocks should not be identical during parameter sharing, because the features of different layers have various scales and statistics. Meanwhile, the ℓ_2-norm of the gradient becomes large and fluctuates across different layers after weight sharing, leading to training instability. Finally, the Central Kernel Alignment (CKA) [29] values, a popular similarity metric, drop significantly in the last few layers, indicating that feature maps generated by the model before and after weight sharing become less correlated, which can be the reason of performance degradation.

In this paper, we propose a new technique, called *weight multiplexing*, to address the above issues. It consists of two components, weight transformation and weight distillation, to jointly compress pre-trained vision transformers. The key idea of *weight transformation* is to impose transformations on the shared weights, such that different layers have slightly different weights, as shown in Fig. 2. This operation can not only promote parameter diversity, but also improve training stability. More concretely, we impose simple linear transformations on the multi-head self-attention (MSA) module and the multilayer perceptron (MLP) module for each weight-shared transformer layer. Each layer includes separate transformation matrices, so the corresponding attention weights and outputs of MLP are different across layers. The layer normalization for different layers is also separated, in contrast to sharing the same parameters. As such, the optimization of weight sharing transformer networks becomes more stable, as demonstrated in Fig. 4.

To mitigate performance degradation, we further equip weight multiplexing with *weight distillation*, such that the information embedded in the pre-trained models can be transferred into the weight-shared small ones, which are much more compact and lightweight. In contrast to previous works that only rely on prediction-level distillation [27, 52], our method additionally considers both attention-level and hidden-state distillation, allowing the smaller model to closely mimic the behavior of the original pre-trained large teacher model.

The experiments demonstrate that our weight multiplexing method achieves clear improvements in accuracy over the baselines and compresses pre-trained vision transformers by 2 times while transferring well to downstream tasks. For instance, with the proposed weight multiplexing, the Mini-Swin-B model with 12-layer parameters obtains 0.8% higher accuracy than the 24-layer Swin-B. Moreover, Mini-DeiT-B with 9M parameters achieves 79.8% top-1 accuracy on ImageNet, being 9.7 times smaller than DeiT-B (with 86 parameters and 81.8% accuracy). The 12M tiny model compressed by our approach transfers well to downstream object detection, achieving an AP of 48.6 on the COCO validation set, which is on par with the original Swin-T using 28M parameters.

We summarize our contributions as follows:

- We systematically investigate the efficacy of weight sharing in vision transformers, and analyze the underlying reasons of issues brought by weight sharing.
- We propose a novel compression framework termed MiniViT for general vision transformers. Experimental results demonstrate that MiniViT can achieve a large compression ratio without losing accuracy. Furthermore, the performance of MiniViT transfers well to downstream benchmarks.

2. Background

Before presenting our method, we first briefly review some background on vision transformers and parameter sharing, which are fundamental to this work.

2.1. Vision Transformers

Transformers, though originally designed for NLP [17, 32, 54], have recently demonstrated their great potentials in computer vision [18, 36, 52]. Vision transformers first split an input image into a sequence of 2D patches known as tokens. They then flatten and transform these patches to D-dimensional vectors using a linear projection [18] or stacked CNN layers [61], also known as patch embeddings. To retain positional information, positional embeddings are added to patch embeddings. The combined embeddings are then fed to a transformer encoder (described below). Lastly, a linear layer is used to produce the final classification.

A transformer encoder consists of alternating blocks of *multihead self-attention* (MSA) and *multi-layer perceptron* (MLP) blocks. Layer normalization (LN) [5] and residual connections are applied before and after each block, respectively. We elaborate on the MSA and MLP blocks as below.
MSA: Let M be the number of heads, also known as self-attention modules. Given the input sequence $\mathbf{Z}_0 \in \mathbb{R}^{N\times D}$, in the k^{th} head, we generate queries, keys, and values by linear projections, denoted by $\mathbf{Q}_k$, $\mathbf{K}_k$, and $\mathbf{V}_k \in \mathbb{R}^{N\times d}$ respectively, where N is the number of tokens. D and d are

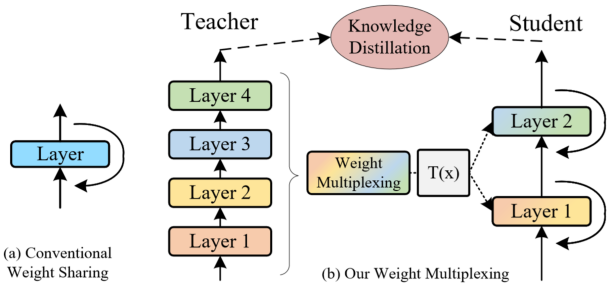


Figure 2. Classical weight sharing versus weight multiplexing.

the dimensions of patch embeddings and *Q-K-V* matrices, respectively. We then compute a weighted sum over all values for each position in the sequence. The weights, called attentions and denoted by $\mathbf{A}_k$, are based on the pairwise similarity between two elements in the sequence, namely

$$\mathbf{h}_k = \mathbf{A}_k \mathbf{V}_k, \text{and} \tag{1}$$

$$\mathbf{A}_k = \text{softmax}\left(\frac{\mathbf{Q}_k \mathbf{K}_k^T}{\sqrt{d}}\right), \tag{2}$$

where *softmax*$(\cdot)$ is conducted on each row of the input matrix. Finally, a fully-connected layer is applied to the concatenation of the outputs of all heads.

MLP: The MLP block comprises two fully-connected layers with an activation function denoted by $\sigma(\cdot)$, usually GELU [24]. Let $\mathbf{Y} \in \mathbb{R}^{N\times d}$ be the input of MLP. The output of MLP can be expressed as

$$\mathbf{H} = \sigma(\mathbf{Y}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)}, \tag{3}$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{d\times d'}$, $\mathbf{b}^{(1)} \in \mathbb{R}^{d'}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{d'\times d}$, and $\mathbf{b}^{(2)} \in \mathbb{R}^{d}$ are the weights and biases for the first and second layers, respectively. Notably, we usually set $d' > d$.

2.2. Weight Sharing

Weight sharing is a simple but effective way to improve parameter efficiency. The core idea is to share parameters across layers, as shown in Fig. 2(a). Mathematically, weight sharing can be formulated as a recursive update of one transformer block f (i.e., one shared layer):

$$\mathbf{Z}_{i+1} = f(\mathbf{Z}_i; \boldsymbol{\theta}), \quad i = 0, \ldots, L-1, \tag{4}$$

where $\mathbf{Z}_i$ denotes the feature embedding of the sequence in layer i, L is the total number of layers, and $\boldsymbol{\theta}$ represents the shared weights of the transformer block across all layers. The efficacy of weight sharing has been explored and proved in natural language transformer models [6, 15, 32]. It can prevent the number of parameters from growing with the depth of the network without seriously hurting the performance, thus improving parameter-efficiency.

3. Method

In this section, we describe our proposed weight multiplexing strategy for vision transformer compression. It consists of two key components, weight transformation and weight distillation, to improve training stability and model performance during weight sharing. Finally, we depict the pipeline of model compression with weight multiplexing.

3.1. Weight Multiplexing

The potential of weight sharing has been demonstrated in NLP [6, 15, 32]; however, its efficacy is still unclear in vision transformers. To examine this, we directly apply the cross-layer weight sharing in Eq. (4) on the DeiT-S [52] and Swin-B [36] transformer models, and observe two issues: training instability and performance degradation. Based on our analysis in Sec. 4.2, the strict identity of weights across different layers is the main cause of the issues. In particular, the ℓ_2-norm of the gradients after weight sharing becomes large and fluctuates in different transformer blocks, as shown in Fig. 4. Furthermore, the CKA values indicate that feature maps generated by the model after weight sharing are less correlated with the original model, as shown in Fig. 5. To solve these issues, inspired by the multiplexing technologies in telecommunications [1, 46], we propose a new technique called *weight multiplexing* for transformer compression. It combines multi-layer weights into a single weight over a shared part, while involving transformation and distillation to increase parameter diversity.

More concretely, as shown in Fig. 2(b), the proposed weight multiplexing method consists of (1) sharing weights across multiple transformer blocks, which can be considered as a combination process in multiplexing [46]; (2) introducing transformations in each layer to mimic demultiplexing [46]; and (3) applying knowledge distillation to increase the similarity of feature representations between the models before and after compression. Following Eq. (4), we can re-formulate our weight multiplexing as follows:

$$\mathbf{Z}_{i+1} = f(\mathbf{Z}_i; \boldsymbol{\theta}, \boldsymbol{\theta}'_i), \; i = 0, \ldots, L-1, \tag{5}$$

where $\boldsymbol{\theta}'_i$ represents the weights of transformation blocks in the i^{th} transformer layer. Note that the number of parameters in $\boldsymbol{\theta}'_i$ is far fewer than $\boldsymbol{\theta}$.

3.1.1 Weight Transformation

The transformations in our method are imposed on both attention matrices and feed-forward networks. Such transformations allow each layer to be different, thus elevating parameter diversity and model representation capability. As illustrated in Fig. 3 (Right), the parameters of transformation kernels are *not* shared across layers, while all the other blocks in the original transformer are shared except LayerNorm [5]. Since the shared blocks occupy the vast majority of the model parameters, the model size only increases slightly after weight multiplexing.

Transformation for MSA. To improve parameter diversity, we insert two linear transformations before and after

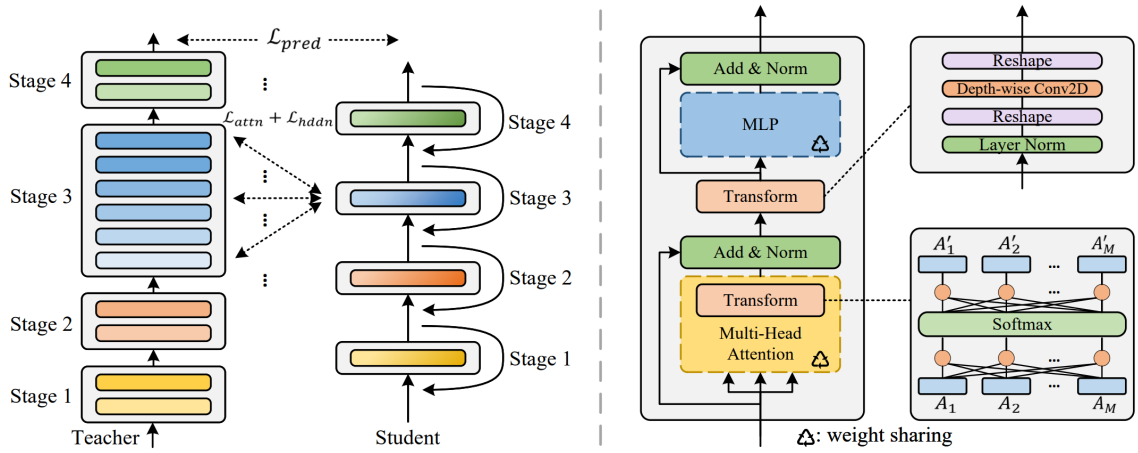


Figure 3. **Left:** The overall framework of MiniViT. Note that the number of stages are configurable, instead of being fixed in Swin transformers [36]. The transformer layers in each stage of the original models to be compressed should have identical structures and dimension. **Right:** The detailed transformer block in a MiniViT. We share weights of MSA and MLP in each stage, and add two transformation blocks to increase the parameter diversity. The transformation blocks and normalization layers are not shared.

the softmax self-attention module, respectively. Formally, different from the original self-attention in Eq. (1-2), the transformation-equipped attention is defined as

$$\mathbf{h}_k = \mathbf{A}'_k \mathbf{V}_k = \sum_{n=1}^{M} \mathbf{F}^{(1)}_{kn} \mathbf{A}_n \mathbf{V}_k, \tag{6}$$

$$\mathbf{A}_n = \text{softmax}\left(\sum_{m=1}^{M} \mathbf{F}^{(2)}_{nm} \frac{\mathbf{Q}_m \mathbf{K}^T_m}{\sqrt{d}} \right), \tag{7}$$

where $\mathbf{F}^{(1)}$, $\mathbf{F}^{(2)} \in \mathbb{R}^{M \times M}$ are the linear transformation kernels before and after the *softmax*, respectively. Such linear transformations can make each attention matrix $\mathbf{A}_n$ different, while combining information across attention heads to increase parameter variance.

Transformation for MLP. On the other hand, we further impose a lightweight transformation for MLP to elevate parameter diversity. In particular, let the input be $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_d]$, where $\mathbf{y}_l$ denotes the l^{th} position of embedding vectors of all tokens. We then introduce d linear transformations to convert $\mathbf{Y}$ into $\mathbf{Y}' = [\mathbf{C}^{(1)}\mathbf{y}_1, \ldots, \mathbf{C}^{(d)}\mathbf{y}_d]$, where $\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(d)} \in \mathbb{R}^{N \times N}$ are independent weight matrices of linear layers. Then Eq. (3) is re-formulated as

$$\mathbf{H} = \sigma(\mathbf{Y}'\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)}. \tag{8}$$

To reduce the number of parameters and introduce locality in the transformations, we resort to depth-wise convolution [13] to sparsify and share weights in each weight matrix, leading to only $K^2 d$ parameters compared to $N^2 d$ parameters ($K << N$), where K is the kernel size of convolution. After the transformations, the outputs of MLP become more diverse, improving the parameter efficacy.

Theoretically, with these transformations, the weight-shared layers can restore the behaviors of the pre-trained models, similar to a demultiplexing process. Then the training instability and performance drop issues can be alleviated, because these issues are not observed in the original models. Similar transformations are also applied to improve the performance of transformers without sharing blocks, such as talking-heads attention [49] and CeiT [60]. However, we extend the ability of transformations to circumvent the drawbacks of weight-sharing methods.

3.1.2 Weight Distillation

To compress the large pre-trained models and address the performance degradation issues induced by weight sharing, we further resort to weight distillation to transfer knowledge from the large models to the small and compact models. We consider three types of distillation for transformer blocks, i.e., prediction-logit distillation, self-attention distillation, and hidden-state distillation.

Prediction-Logit Distillation. Hinton et al. [25] firstly demonstrated that deep learning models can achieve better performance by imitating the output behavior of well-performing teacher models during training. We leverage this idea to introduce a prediction loss, as follows:

$$\mathcal{L}_{pred} = CE\left(\text{softmax}\left(\frac{\mathbf{z}_s}{T}\right), \text{softmax}\left(\frac{\mathbf{z}_t}{T}\right)\right), \tag{9}$$

where $\mathbf{z}_s$ and $\mathbf{z}_t$ are the logits predicted by the student and teacher models, respectively, and T is a temperature value which controls the smoothness of logits. In our experiments, we set $T = 1$. CE denotes the cross-entropy loss.

Self-Attention Distillation. Recent literature has shown that utilizing attention maps in transformer layers to guide the training of student models is beneficial [28, 32, 50]. To solve the dimension inconsistency between the student and teacher models due to differing numbers of heads, and inspired by [55], we apply cross-entropy losses on relations among queries, keys, and values in MSA.

In particular, we first append matrices over all heads. For example, we define $\mathbf{Q} = [\mathbf{Q}_1, \ldots, \mathbf{Q}_M] \in \mathbb{R}^{N \times Md}$, and $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times Md}$ in the same way. For notational simplic-

ity, we write $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$ to denote $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ respectively. Then we can generate nine different relation matrices defined by $\mathbf{R}_{ij} = \text{softmax}(\mathbf{S}_i\mathbf{S}_j^T/\sqrt{Md})$. Note that $\mathbf{R}_{12}$ is the attention matrix $\mathbf{A}$. The self-attention distillation loss can be expressed as

$$\mathcal{L}_{attn} = \frac{1}{9N}\sum_{n=1}^{N}\sum_{\substack{i,j\in\\ \{1,2,3\}}} CE(\mathbf{R}^s_{ij,n}, \mathbf{R}^t_{ij,n}), \tag{10}$$

where $\mathbf{R}_{ij,n}$ represents the n^{th} row of $\mathbf{R}_{ij}$.

Hidden-State Distillation. Similarly, we can generate relation matrices for hidden states, *i.e.*, the features output by MLP. Denoting the hidden states of a transformer layer by $\mathbf{H} \in \mathbb{R}^{N\times d}$, the hidden-state distillation loss based on relation matrices is defined as

$$\mathcal{L}_{hddn} = \frac{1}{N}\sum_{n=1}^{N} CE(\mathbf{R}^s_{H,n}, \mathbf{R}^t_{H,n}), \tag{11}$$

where $\mathbf{R}_{H,n}$ indicates the n^{th} row of $\mathbf{R}_H$, which is computed by $\mathbf{R}_H = \text{softmax}(\mathbf{H}\mathbf{H}^T/\sqrt{d})$.

Based on our observation that only using prediction soft labels can yield better performance than using both the prediction and ground truth labels (see the ablation in Sec. 4.2), thus the final distillation objective function is formulated as

$$\mathcal{L}_{train} = \mathcal{L}_{pred} + \beta\mathcal{L}_{attn} + \gamma\mathcal{L}_{hddn}, \tag{12}$$

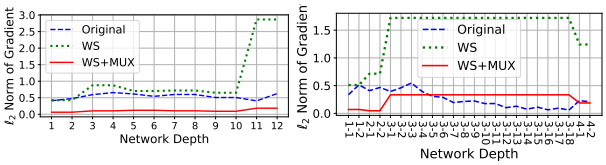
where β and γ are hyperparameters with default values of 1 and 0.1 respectively, unless otherwise specified.

3.2. Compression Pipeline

Our compression pipeline includes two phases:

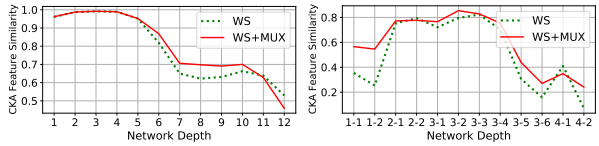
Phase 1: Generating compact architectures with weight transformation. Given a large pre-trained vision transformer model, we first share the parameters across every-K adjacent transformer layers except the LayerNorm [5]. Then we apply weight transformation to each layer by inserting a tiny linear layer before and after the *softmax* layer, as defined in Eq. (6-7). Furthermore, we introduce a depth-wise convolutional layer for MLP. These linear layers and transformation blocks are not shared.

Phase 2: Training the compressed models with weight distillation. In this step, we apply the proposed weight distillation method to transfer knowledge from the large pre-trained models to small ones, using the objective function defined in Eq. (12). Such distillation inside the transformer modules allows the student network to reproduce the behaviors of the teacher network [28], thus extracting more useful knowledge from the large-scale pre-trained models. Note that it is only performed when both the teacher and student models are transformer architectures. In other cases where the architectures of the student and teacher are heterogeneous, we only preserve the prediction-logit distillation.



(a) DeiT-S (b) Swin-B

Figure 4. Comparisons of ℓ_2-norm of gradients during training among the models with weight sharing (WS), and with both weight sharing and multiplexing (WS+MUX), and the original one.



(a) DeiT-S (b) Swin-B

Figure 5. Comparisons of feature similarity with respect to the original model by CKA [29] between the models with weight sharing (WS), and with weight sharing and multiplexing (WS+MUX).

4. Experiments

In this section, we first provide an analysis of weight sharing in vision transformers, followed by experiments on the effects of the proposed weight transformation and weight distillation methods. Next, we show the parameter efficiency of our method by comparing with other state-of-the-art models. Finally, we demonstrate the transferability of the compression models on downstream tasks.

4.1. Implementation Details

Architectures for Compression. In the case of hierarchical models such as Swin [36], we only share parameters of transformer layers in each stage, due to the unaligned parameter dimension in different stages (caused by feature down-sampling), and generate a series of compact models named *Mini-Swins*. For DeiT, one of the popular isomorphic models, we consider it as a single-stage vision transformer and create *Mini-DeiTs*. We make several modifications on DeiT: First, we remove the [class] token. The model is attached with a global average pooling layer and a fully-connected layer for image classification. We also utilize relative position encoding to introduce inductive bias to boost the model convergence [52, 59]. Finally, based on our observation that transformation for FFN only brings limited performance gains in DeiT, we remove the block to speed up both training and inference. The results of MiniViT based on the original DeiT [52] are presented in the *supplementary materials* for comprehensive comparisons.

Training Settings. We train our models from scratch on ImageNet-1K [16] by directly inheriting the hyper-parameters from DeiT [52] or Swin transformers [36] except for the drop path rate, which are set to be 0.0/0.0/0.1 for DeiT-Ti/S/B and 0.0/0.1/0.2 for Swin-T/S/B, respectively. The data augmentation techniques include RandAugment [14], Cutmix [62], Mixup [63], and random erasing. RepeatAug [26] is only applied in DeiT [52]. In the down-

Model	#	WS	MUX		#Params	Top-1	Top-5
			WD	WT		Acc(%)	Acc(%)
Swin-T	1				28M	81.2	95.5
	2	✓			12M	79.0	94.4
	3	✓	✓		12M	79.8	95.1
	4	✓		✓	12M	79.2	94.3
	5	✓	✓	✓	12M	81.4	95.8
DeiT-S	6				22M	79.9	95.0
	7	✓			11M	78.3	94.4
	8	✓	✓		11M	80.1	95.1
	9	✓		✓	11M	79.3	94.8
	10	✓	✓	✓	11M	80.7	95.4

Table 1. Component-wise analysis on ImageNet-1K [16]. WS: Weight Sharing, WD: Weight Distillation, WT: Weight Transformation, MUX: Weight Multiplexing including both WD and WT.

Sharing	Swin-T			DeiT-B		
Strategy	#Params	Top-1	Top-5	#Params	Top-1	Top-5
original	28M	81.2	95.5	86M	81.8	95.6
every-2	16M(1.8×)	82.2	96.2	44M(2.0×)	83.2	96.5
all layers	12M(2.3×)	81.4	95.8	9M(9.7×)	79.8	94.9

Table 2. Ablation study on the number of sharing blocks on Imagenet-1K [16]. Fisrt row: the original model. Second row: sharing two consecutive layers. Third-row: sharing all layers of DeiT [52] and in each stage of Swin [36].

stream tasks, we fine-tune the models for 30 epochs at 384^2 resolution. Specifically, the position encoding is resized with bicubic interpolation. The AdamW [37] optimizer is applied with weight decay 10^{-8} and a cosine scheduler, batch size 256. The learning rates are 2.5×10^{-6} and 10^{-5} for DeiT and Swin, respectively. All models are implemented using PyTorch [43] and Timm library [57], and trained for 300 epochs with 8 NVIDIA Tesla V100 GPUs.

For compressing Swin [36], we adopt the ImageNet-22k [16] pre-trained Swin-B with 88M parameters as the teacher model. Thus, our compressed models can still learn the knowledge of the large-scale ImageNet-22k data by distillation without requiring the access to the dataset. Similarly, for DeiT [52], we use RegNet-16GF [44] with 84M parameters as the teacher, and only perform prediction-logit distillation due to heterogenous architectures between CNN and ViT. We share the consecutive two layers except Swin-T where all layers in each stage are shared. For the depth-wise convolution in weight transformation, we set the kernel size to be the same as the window size in Swin [36] and the stride as 1. Besides, same-padding is used.

4.2. Analysis and Ablation

Analysis of Weight Sharing (WS). When directly applying weight sharing as in Eq. (4), the training process collapses, and the model suffers from severe performance drop.

First, we investigate the training stability. We share every two layers in DeiT-S while sharing all layers in each stage of Swin-B. As shown in Fig. 4, the ℓ_2-norm of gradients in DeiT-S and Swin-B after weight sharing becomes much larger, indicating a fast change of magnitude of weights. Furthermore, weight sharing causes fluctuations in the gradient norm across different layers. This may lead to different optimization paces of layers. In particular, some layers

Model	GT	$\mathcal{L}_{pred}$	$\mathcal{L}_{attn}$	$\mathcal{L}_{hddn}$	Top-1	Top-5
Swin-B (22k) (Teacher)	✓				85.2	97.5
Mini-Swin-T w/o distillation	✓				79.2	94.3
Mini-Swin-T		✓			81.2	95.6
	✓	✓			80.9	95.5
		✓	✓		81.4	95.7
		✓		✓	81.5	95.8
		✓	✓	✓	81.4	95.8

Table 3. Ablation study on different distillation losses on ImageNet-1K [16]. We use Swin-B [36] pre-trained on ImageNet-22K [16] as the teacher model. The weights for all losses are 1 except using both GT and $\mathcal{L}_{pred}$, where both weights are 0.5.

are updated quickly, whereas the other parts are hardly optimized, making the model likely to converge to a bad local optimum or even diverge in training. Therefore, strictly identical weights shared across layers lead to training instability. However, our weight multiplexing method can both reduce the gradient norm and increase the smoothness across layers by introducing transformations to improve parameter diversity, promoting a more stable training process.

As for performance analysis, we present a comparison of feature similarities with CKA between weight sharing and weight multiplexing in Fig. 5. Higher CKA values indicate more similar feature representations between two models [29], thus achieving similar performance. We observe that both DeiT and Swin suffer from large deviations of feature representations after applying weight sharing, especially in the last several layers, which may be one of the reasons for performance drops brought about by weight sharing. Nevertheless, our proposed weight multiplexing method can improve the similarities.

Component-wise Analysis. We evaluate the effects of different components in our proposed weight multiplexing method on ImageNet-1K [16], and report the results in Tab. 1. Our baselines are the official Swin-T [36] and DeiT-S [52] models. After applying the weight-sharing method, the numbers of parameters of both models are halved, whereas the accuracy also decreases by 2% (#1 *vs.* #2, #6 *vs.* #7). The performance can be improved by either applying weight distillation (#2 *vs.* #3, #7 *vs.* #8) or weight transformation (#2 *vs.* #4, #7 *vs.* #9), which demonstrates their individual effectiveness. It is notable that weight transformation only introduces a few parameters. Furthermore, when combining all three components, MiniViTs can achieve the best accuracy (#5 and #10), which even outperforms the original Swin-T and DeiT-S models.

Number of Sharing Blocks. We study the performance of MiniViTs in different sharing settings. In Tab. 2, sharing every two blocks in each stage of Swin-T or DeiT-B can significantly reduce the number of parameters from 28M to 16M, and 86M to 44M, respectively, whereas the Top-1 accuracy becomes 1% higher. Note that we consider DeiT as a single-stage vision transformer. In the extreme case where all blocks in each stage are shared, Mini-Swin-T can still outperform the original model with only 43% of the parameters. Mini-DeiT-B can achieve 90% parameter reduction

Model	#Param. (M)	MACs (B)	IN-1k Acc (%)	IN-Real Acc (%)	IN-V2 Acc (%)	Input
Convnets						
ResNet-50 [23, 57]	25	4.1	69.8	77.3	57.1	224^2
RegNetY-16GF [44]	84	15.9	82.9	88.1	72.4	224^2
EfficientNet-B1 [51]	8	0.7	79.1	84.9	66.9	240^2
EfficientNet-B5 [51]	30	9.9	83.6	88.3	73.6	456^2
Transformers						
ViT-B/16 [18]	86	55.6	77.9	83.6	-	384^2
ViT-L/16 [18]	307	191.5	76.5	82.2	-	384^2
VTP (20%) [64]	67	13.8	81.3	-	-	224^2
VTP (40%) [64]	48	10.0	80.7	-	-	224^2
AutoFormer-T [11]	6	1.3	74.7	-	-	224^2
AutoFormer-S [11]	23	5.1	81.7	-	-	224^2
AutoFormer-B [11]	54	11.0	82.4	-	-	224^2
S^2ViTE-T [12]	4	1.0	70.1	-	-	224^2
S^2ViTE-S [12]	15	3.1	79.2	-	-	224^2
S^2ViTE-B [12]	57	11.8	82.2	-	-	224^2
DeiT-Ti [52]	5	1.3	72.2	80.1	60.4	224^2
DeiT-S [52]	22	4.6	79.9	85.7	68.5	224^2
DeiT-B [52]	86	17.6	81.8	86.7	71.5	224^2
DeiT-B↑384 [52]	87	55.6	82.9	87.7	72.4	384^2
DeiT-B⚗↑384 [52]	88	55.7	84.5	89.0	74.8	384^2
Mini-DeiT-Ti **(ours)**	3(1.7×)	1.3	72.8(+0.6)	83.5(+3.4)	61.3(+0.9)	224^2
Mini-DeiT-S **(ours)**	11(2.0×)	4.7	80.7(+0.8)	88.4(+2.7)	69.5(+1.0)	224^2
Mini-DeiT-B **(ours)**	44(2.0×)	17.7	83.2(+1.4)	89.6(+2.9)	73.0(+1.5)	224^2
Mini-DeiT-B↑384 **(ours)**	44(2.0×)	56.9	84.7(+1.8)	89.9(+2.2)	75.2(+2.8)	384^2
Swin-B (22k) [36]	88	15.4	85.2	89.2	75.3	224^2
Swin-B↑384 (22k) [36]	88	47.1	86.4	90.0	76.6	384^2
Swin-T [36]	28	4.5	81.2	86.6	69.6	224^2
Swin-S [36]	50	8.7	83.2	87.6	71.9	224^2
Swin-B [36]	88	15.4	83.5	87.8	72.5	224^2
Swin-B↑384 [36]	88	47.1	84.5	88.6	73.2	384^2
Mini-Swin-T **(ours)**	12(2.3×)	4.6	81.4(+0.2)	87.1(+0.5)	70.5(+0.9)	224^2
Mini-Swin-S **(ours)**	26(1.9×)	8.9	83.6(+0.4)	88.7(+1.1)	73.8(+1.9)	224^2
Mini-Swin-B **(ours)**	46(1.9×)	15.7	84.3(+0.8)	89.0(+1.2)	74.4(+1.9)	224^2
Mini-Swin-B↑384 **(ours)**	47(1.9×)	49.4	85.5(+1.0)	89.5(+0.9)	76.1(+2.9)	384^2

Table 4. MiniViT Top-1 accuracy on ImageNet-1K [16], Real [7] and V2 [47] with comparisons to state-of-the-art models. Our MiniViTs consistently outperform existing transformer-based visual models and CNNs with fewer parameters. ↑ denotes fine-tuning with 384^2 resolution.

with only a 2% performance drop. The results indicate that parameter redundancy exists in vision transformers and our proposed MiniViT is effective in improving parameter efficiency. Moreover, MiniViT is configurable to satisfy various requirements on model size and performance.

Distillation Losses. We investigate the effectiveness of different types of distillation in MiniViT. When using both ground truth (GT) and prediction soft labels as loss, we set the trade-off weights to 0.5. We use Swin-T as our baseline. As shown in Tab. 3, compared to only using the prediction loss, the extra GT labels leads to a 0.3% performance drop for Swin, which is due to the degradation of learning ability brought by weight sharing. Furthermore, we also observe around a 0.2% improvement in accuracy after applying the attention-level and hidden-state distillation, indicating the effectiveness of our proposed distillation method.

4.3. Results on ImageNet

We compare our proposed MiniViT models for diverse parameter sizes with the state-of-the-art ones on ImageNet-1K [16], Real [7] and V2 [47]. The top-1 accuracy is reported in Tab. 4. Note that our MiniViTs are trained from scratch on ImageNet-1K only without using the large-scale ImageNet-22K dataset. It is clear that the Mini-Swin and Mini-DeiT model families, compressed over Swin transformers [36] and DeiT [52], respectively, achieve accuracy improvements with only half as many parameters. In particular, using 46M parameters, our Mini-Swin-B performs 0.8% accuracy higher than Swin B on ImageNet-1K. Besides, our Mini-DeiT-B reduces 50% parameter amount and achieves a 1.8% performance improvement. Moreover, on ImageNet-Real, our Mini-DeiTs achieve 2% better performance than DeiTs, while Mini-Swins are also superior to Swins by around 1%. On ImageNet-V2, MiniViTs can outperform the original models up to 3% (Base↑384).

Compared to other efficient vision transformer methods, MiniViT is also competitive. Specifically, Mini-DeiT-B, with only 44M parameters, achieves 1.0% and 2.5% higher Top-1 accuracy than S^2ViTE-B (57M) [12], and VTP (40%) (48M) [64], respectively.

Our Mini-DeiTs can also outperform automatical nearul architecture search methods. In particular, Mini-DeiT-B uses only 44M parameters to achieve 0.8% better accuracy than AutoFormer-B (54M) [11]. Our tiny model, Mini-DeiT-Ti, still has comparable performance to AutoFormer-T, while the model size is merely 3M.

Model	#Params	ImageNet1k	CIFAR-10	CIFAR-100	Flowers	Cars	Pets
Grafit ResNet-50 [53]	25M	79.6	-	-	98.2	92.5	-
EfficientNet-B5 [51]	30M	83.6	98.7	91.1	98.5	-	-
EfficientNet-B7 [51]	66M	84.3	98.9	91.7	98.8	94.7	-
ViT-B/32 [18]	86M	73.4	97.8	86.3	85.4	-	92.0
ViT-B/16 [18]	86M	77.9	98.1	87.1	89.5	-	93.8
NViT-T [3]	6.4M	73.9	98.2	85.7	-	-	-
NVP-T [3]	6.9M	76.2	98.3	85.9	-	-	-
DeiT-B [52]	86M	81.8	99.1	90.8	98.4	92.1	-
DeiT-B↑384 [52]	87M	83.1	99.1	90.8	98.5	93.3	-
DeiT-B⚗ [52]	87M	83.4	99.1	91.3	98.8	92.9	-
DeiT-B⚗↑384 [52]	88M	84.4	99.2	91.4	98.9	93.9	-
Mini-DeiT-B↑384	44M	84.7	99.3	91.5	98.3	93.8	95.5

Table 5. MiniViT results on downstream classification datasets.

4.4. Transfer Learning Results

Image Classification. We transfer MiniViT to a collection of commonly used recognition datasets: (1) CIFAR-10 and CIFAR-100 [31]; (2) fine-grained classification: Flowers [39], Stanford Cars [2], and Oxford-IIIT Pets [42]. Following the fine-tuning settings of DeiT [52], an SGD optimizer is adopted with learning rate 5×10^{-3}, batch size 256, weight decay 10^{-4}, and disabled random erase, except Cars [2], using AdamW [37] with learning rate 10^{-3}, weight decay 5×10^{-2} and random erase. We train models for 300 epochs on Flowers and Cars, and 1000 on others. Tab. 5 shows the results of Top-1 accuracy. Compared to the state-of-the-art ConvNets and transformer-based models, Mini-DeiT-B↑384 achieves comparable or even better results on all datasets, only using 44M parameters.

Object Detection. We also investigate the transferabil-

#	Backbone	#Params	WT	WD	Det KD	ImageNet Top1-Acc	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
1	Swin-T [36]	28M				81.2	48.1	67.1	52.1	31.1	51.2	63.5
2	Mini-Swin-T	12M				79.0	46.5	65.5	50.6	29.9	50.0	61.6
3	Mini-Swin-T	12M	✓			79.2	47.5	66.1	51.8	30.8	50.6	62.2
4	Mini-Swin-T	12M	✓	✓		81.4	47.8	66.5	51.9	30.4	51.3	63.2
5	Mini-Swin-T	12M	✓	✓	✓	81.4	48.6	67.2	52.6	31.0	52.4	64.2

Table 6. Comparison on COCO [35] object detection using Cascade Mask R-CNN [8, 22]. We replace the original backbone with our compressed models, and report the number of parameters of the backbone. We train detectors for 12 epochs.

ity of MiniViT to the COCO 2017 detection dataset [35]. We use Cascade R-CNN [8] with Swin-T [36] as our baseline. For distillation during fine-tuning (Det KD), Cascade R-CNN with Swin-B is adopted as the teacher model, and a mean square error (MSE) loss on the backbone outputs of the student and teacher models is utilized with weight 0.1. We follow the same training techniques with Swin-based Cascade R-CNN [36]. As shown in Tab. 6, (1) weight-sharing can cause a 1.6 AP decrease, although it reduces the number of parameter; (2) models with weight transformation can transfer well, with only a 0.6 AP decrease but fewer parameters; (3) MiniViT, combining weight sharing, transformation, and distillation, can achieve comparable AP performance to the baseline with 57% parameter reduction on the backbone, and can achieve the best result after coupled with detection distillation in the fine-tuning stage.

5. Related Work

Vision Transformer. Transformers were originally proposed for language modeling [54], and recently applied in computer vision. It has shown promising potential on a variety of tasks, such as recognition, detection, and segmentation [9, 18, 34]. Dosovitskiy et al. first introduced the ViT model [18], a pure transformer architecture for visual recognition pre-trained on large-scale data. This work inspired a large amount of follow-up approaches [10, 21, 36, 52, 61]. Among them, DeiT [52] and the Swin transformer [36] are two representative ones. DeiT [52] demonstrates that large-scale data is not necessary when training a ViT model. Swin transformers [36] introduce a hierarchical structure to the ViT regime, mimicking traditional convolutional networks. Equipped with shifted windows, they have shown promising results on visual recognition and downstream tasks.

ViT models are becoming increasingly heavy and expensive, so several recent works have proposed methods for compression. One primitive way is pruning, such as removing redundant tokens [41, 45], attention heads [12], or hidden dimensions [3, 64]. A recent work [4] combines pruning, skipping, and distillation together and proposes a unified compression framework for vision transformers. However, most existing compression literature focuses on isomorphic vision transformers, whereas their efficacy on hierarchical vision transformers remains unclear. Our method is more general, aiming at the compression of both isomorphic and hierarchical vision transformers.

Weight Sharing. The history of weight sharing can be traced back to the 1990s [33, 40, 48]. Due to its good capability in improving parameter efficiency, weight sharing has been adopted in transformers for language tasks [20, 32, 38]. Universal transformers [15] propose a weight sharing mechanism across both positions and time steps, yielding better performance than standard transformers [54] on a variety of sequence-to-sequence tasks. After observing that the output of weight-shared models converges to a fixed point, Deep Equilibrium Models [6] design a Quasi-Newton method utilizing equilibrium states to fit transformers, and have been demonstrated to outperform other deep sequence models on language tasks. Different from the previous work, Albert [32] found a performance drop in NLP benchmarks after sharing all weights of transformers in BERT [17]. By contrast, our work investigates the efficacy of weight sharing in vision transformers, while equipping it with transformation and distillation to further enhance the method.

Knowledge Distillation. Distillation in a teacher-student framework, is widely used to reduce model sizes. It has been extensively studied in convolutional networks [19]. However, in vision transformers, it is still under-explored. A few relevant recent works include Touvron et al. [52] who introduce a distillation token to allow the transformer to learn from a ConvNet teacher, and Jia et al. [27] propose to excavate knowledge from the teacher transformer via the connection between images and patches. Distillation inside the transformer (e.g., MSA and MLP) has been verified to be effective in transformers for NLP [28, 50, 55, 56]. In contrast, our work provides an initial design for attention-level and hidden-state distillation, and explores their effectiveness in vision transformer compression.

6. Conclusion

We have proposed a new compression framework, *i.e.* MiniViT, for vision transformers. Our method combines weight sharing, transformation, and distillation to reduce the number of parameters while achieving even better performance compared to the original models.

Limitations. One limitation of MiniViT is that, despite improving the parameter efficiency, the computational cost is slightly increased compared to the classical weight sharing strategy, due to the introduced weight transformation blocks. Second, we observe that MiniViT suffers from moderate performance degradation as the compression ratio increases. In future work, we are interested in further improving both the parameter and computational efficiency.

Broader Impacts. Presented in the supplementary materials.

Acknowledgement. We would like to thank Microsoft NNI and OpenPAI v-team for kind helps and supports. Thanks to Prof. Po-Ling Loh for the final proofreading.

References

[1] Multiplexing. https://en.wikipedia.org/wiki/Multiplexing. 3

[2] 3d object representations for fine-grained categorization. In *3dRR*, 2013. 7

[3] Anonymous authors. Nvit: Vision transformer compression and parameter redistribution. In *OpenView*, 2021. 7, 8

[4] Anonymous authors. Unified visual transformer compression. In *OpenView*, 2021. 8

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 2, 3, 5

[6] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *NeurIPS*, 2019. 3, 8

[7] Lucas Beyer, Olivier J Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are we done with imagenet? *arXiv preprint arXiv:2006.07159*, 2020. 7

[8] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *CVPR*, 2018. 8

[9] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 8

[10] Chun-Fu Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. *ICCV*, 2021. 8

[11] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *ICCV*, 2021. 7

[12] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. In *NeurIPS*, 2021. 7, 8

[13] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 4

[14] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR*, 2020. 5

[15] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *ICLR*, 2019. 3, 8

[16] 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. 1, 5, 6, 7

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, 2019. 2, 8

[18] 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, 7, 8

[19] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *IJCV*, 2021. 8

[20] Jiatao Gu, Changhan Wang, and Junbo Zhao. Levenshtein transformer. In *NeurIPS*, 2019. 8

[21] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *NeurIPS*, 2021. 8

[22] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. 8

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

[24] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 3

[25] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 4

[26] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *CVPR*, June 2020. 5

[27] 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. 2, 8

[28] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. In *EMNLP*, 2020. 4, 5, 8

[29] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019. 2, 5, 6

[30] Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. Revealing the dark secrets of bert. In *EMNLP*, 2019. 1

[31] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 7

[32] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *ICLR*, 2020. 1, 2, 3, 4, 8

[33] Yann LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, 1989. 1, 8

[34] Justin Liang, Namdar Homayounfar, Wei-Chiu Ma, Yuwen Xiong, Rui Hu, and Raquel Urtasun. Polytransform: Deep polygon transformer for instance segmentation. In *CVPR*, 2020. 8

[35] 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

[36] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021. 1, 2, 3, 4, 5, 6, 7, 8

[37] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2018. 6, 7

[38] Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song. A tensorized transformer for language modeling. In *NeurIPS*, 2019. 8

[39] M-E Nilsback and Andrew Zisserman. A visual vocabulary for flower classification. In *CVPR*, 2006. 7

[40] Steven J Nowlan and Geoffrey E Hinton. Simplifying neural networks by soft weight-sharing. *Neural computation*, 1992. 1, 8

[41] Bowen Pan, Rameswar Panda, Yifan Jiang, Zhangyang Wang, Rogerio Feris, and Aude Oliva. Ia-red2: Interpretability-aware redundancy reduction for vision transformers. In *NeurIPS*, 2021. 8

[42] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *CVPR*, 2012. 7

[43] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019. 6

[44] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, 2020. 6, 7

[45] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *NeurIPS*, 2021. 8

[46] Theodore S Rappaport et al. *Wireless communications: principles and practice*, volume 2. prentice hall PTR New Jersey, 1996. 3

[47] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *ICML*, 2019. 7

[48] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 1986. 8

[49] Noam Shazeer, Zhenzhong Lan, Youlong Cheng, Nan Ding, and Le Hou. Talking-heads attention. *arXiv preprint arXiv:2003.02436*, 2020. 4

[50] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *ACL*, 2020. 4, 8

[51] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 7

[52] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*. PMLR, 2021. 1, 2, 3, 5, 6, 7, 8

[53] Hugo Touvron, Alexandre Sablayrolles, Matthijs Douze, Matthieu Cord, and Hervé Jégou. Grafit: Learning fine-grained image representations with coarse labels. In *ICCV*, 2021. 7

[54] 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. 2, 8

[55] Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. Minilmv2: Multi-head self-attention relation distillation for compressing pretrained transformers. *ACL*, 2021. 4, 8

[56] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *NeurIPS*, 2020. 8

[57] Ross Wightman. Pytorch image models, 2019. 6, 7

[58] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. *ICCV*, 2021. 1

[59] Kan Wu, Houwen Peng, Minghao Chen, Jianlong Fu, and Hongyang Chao. Rethinking and improving relative position encoding for vision transformer. In *ICCV*, 2021. 5

[60] Kun Yuan, Shaopeng Guo, Ziwei Liu, Aojun Zhou, Fengwei Yu, and Wei Wu. Incorporating convolution designs into visual transformers. *ICCV*, 2021. 4

[61] 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. *ICCV*, 2021. 2, 8

[62] 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. 5

[63] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 5

[64] Mingjian Zhu, Yehui Tang, and Kai Han. Vision transformer pruning. In *KDD Workshop on Model Mining*, 2021. 7, 8