A General and Efficient Training for Transformer via Token Expansion

Wenxuan Huang[1*]   Yunhang Shen[2*]   Jiao Xie[3]   Baochang Zhang[4]   Gaoqi He[1]

Ke Li[2]   Xing Sun[2]   Shaohui Lin[1,5✉]

[1]School of Computer Science and Technology, East China Normal University, Shanghai, China

[2]Tencent Youtu Lab, China   [3]Xiamen University, China   [4]Beihang University, China

[5]Key Laboratory of Advanced Theory and Application in Statistics and Data Science - MOE, China

osilly0616@gmail.com, shenyunhang01@gmail.com, jiaoxie1990@126.com, bczhang@buaa.edu.cn

gqhe@cs.ecnu.edu.cn, tristanli.sh@gmail.com, winfred.sun@gmail.com, shaohuilin007@gmail.com

Abstract

The remarkable performance of Vision Transformers (ViTs) typically requires an extremely large training cost. Existing methods have attempted to accelerate the training of ViTs, yet typically disregard method universality with accuracy dropping. Meanwhile, they break the training consistency of the original transformers, including the consistency of hyper-parameters, architecture, and strategy, which prevents them from being widely applied to different Transformer networks. In this paper, we propose a novel token growth scheme Token Expansion (termed ToE) to achieve consistent training acceleration for ViTs. We introduce an "initialization-expansion-merging" pipeline to maintain the integrity of the intermediate feature distribution of original transformers, preventing the loss of crucial learnable information in the training process. ToE can not only be seamlessly integrated into the training and fine-tuning process of transformers (e.g., DeiT and LV-ViT), but also effective for efficient training frameworks (e.g., EfficientTrain), without twisting the original training hyper-parameters, architecture, and introducing additional training strategies. Extensive experiments demonstrate that ToE achieves about $1.3\times$ faster for the training of ViTs in a lossless manner, or even with performance gains over the full-token training baselines. Code is available at https://github.com/Osilly/TokenExpansion.

1. Introduction

Transformers have achieved excellent performance in the tasks of natural language processing (NLP) [1–3] and computer vision [4–7]. Despite their great success, modern Transformer models typically require extremely large parameters and computation consumption due to the quadratic com-

*Equal contribution.   ✉Corresponding author.

Table 1. Training results for DeiT [4] on ImageNet-1K. DeiT does *not* use the EMA strategy by default. a/b in the column of Top-1 Acc. means without/with EMA strategy using the official GitHub repo. The training time is averagely measured on one/four NVIDIA RTX A6000 GPUs 3 times with a batch size of $1,024$ for DeiT-Tiny/Base, respectively.

Model	Method	Training consistency			Top-1 Acc. (%)	Training time
		Hyper	Arch	Strategy		(GPU hours)
Tiny	Baseline [4]	–	–	–	72.2	54.6h
	S²ViTE (600 epoch) [10]	×	×	✓	70.1 (-2.1)	–
	ToMe$_{rs}^{DeiT}$ [11]	✓	×	✓	71.7 (-0.5)	53.3h
	NetworkExpansion$_{6\rightarrow12}$ [12]	✓	✓	×	70.3 (-1.9) / 70.1 (-2.1)	43.2h
	ToE$_{r_1=0.5}$ (Ours)	✓	✓	✓	**72.6 (+0.4)**	44.2h
Base	Baseline [4]	–	–	–	81.8	292.8h
	StackBERT [13]	✓	✓	×	80.8 (-1.0)	231.6h
	NetworkExpansion$_{6\rightarrow12}$ [12]	✓	✓	×	81.0 (-0.8) / 81.5 (-0.3)	226.8h
	ToE$_{r_1=0.5}$ (Ours)	✓	✓	✓	**81.6 (-0.2)**	231.2h

putational complexity in the self-attention module. For example, ViT-H/14 [8] requires $\sim$1,000B FLOPs, which is $250\times$ larger than ResNet-50 [9]. The entire training process needs a significant amount of computing resources to reach model convergence, resulting in a substantial computation overhead. To reduce the computational cost of large models, there has been growing research attention on accelerating Transformers for either training or inference.

Existing Transformer pruning methods [14–22] aim to reduce the inference complexity. Among them, structure pruning [14–17] and token pruning [18–22] focus on reducing the neurons or tokens of Transformers to accelerate the inference. However, these pruning methods require additional training computational cost in each forward-backward iteration to determine which neurons or tokens are important enough to be retained, or the fine-tuning for pruned models. Recently, Transformer quantization [23–26] accelerates the inference via low-bit computation, but they also cannot reduce the training computation cost. Thus, it is challenging for them to effectively accelerate the training of Transformers in practical scenarios, *e.g.*, cloud service.

To reduce the training computation overhead, recent works [12, 13, 27–29] have proposed structure growth methods. They update a smaller number of model parameters during the early stages of training and gradually increase
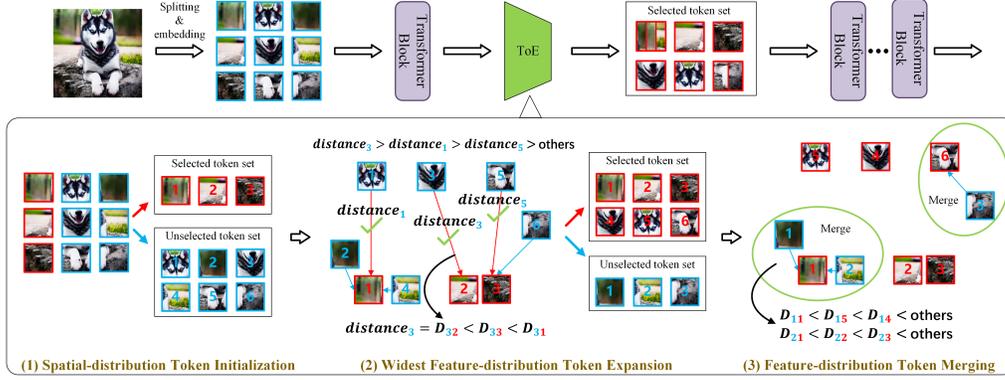
Figure 1. The "initialization-expansion-merging" pipeline of proposed ToE. We take the 1st training stage ($\delta = 1$), the kept rate $r_1 = 2r_0 = \frac{2}{3}$, the repetition step $k = 1$ as example. ToE is only added after the first Transformer block to guide the token selection and usage. During training, steps (1), (2), and (3) are performed for each iteration with the reduction of token numbers. First, seed tokens are selected for token initialization through step (1). Then, the number of tokens is expanded via step (2) for token expansion. Finally, we merge the unselected token set (blue boxes) into the selected one (red boxes) with the close feature distributions in step (3) for token merging. During testing, ToE can be safely removed to generate the same Transformer architecture as the original full-token Transformer.

the number of parameters involved in the updating process as training progresses. However, the existing methods fail to achieve general transformer training acceleration without accuracy dropping (shown in Tab. 1), and they break the *training consistency* of the original transformers from three perspectives: (1) Hyper-parameter consistency. Existing methods (*e.g.*, SViTE [10]) delicately tune training hyper-parameters (*e.g.*, learning rate and epoch number) of the original models, which are sensitive to individual ViTs [4] and require additional trial-and-error costs for different networks. (2) Architecture consistency. Existing methods [10, 11] alter the final model architectures, which may deviate from the user's requirements and potentially necessitates additional hardware/software support to implement real training speedup. For example, ToMe [11] progressively merges similar tokens layer-by-layer to reduce the number of tokens in ViTs during training, which replaces the attention operators with the weighted average attention modules, generating a different model architecture that deviates from the original Transformer. Moreover, it cannot significantly accelerate the practical training due to the unfriendly computation. (3) Strategy consistency. Existing methods [12, 13, 27] may suffer from performance deterioration across different Transformers by adding additional training strategies, such as EMA and reset optimizer states. It means the effectiveness of these strategies is for specific models, which limits the method's universality whether employing them for training. In Tab. 1, the extra EMA strategy in [12] plays different roles to the performance across different models, *i.e.*, the effectiveness for DeiT-base but not for DeiT-tiny. Thus, this begs our rethinkings: *How to implement real and friendly training speedup for Transformers while keeping the training consistency and high accuracy?*

To answer the above question, we propose a novel token growth scheme, *Token Expansion* (termed **ToE**) to achieve general training acceleration for ViTs, while adhering to

the training consistency of original models. Specifically, we present an "initialization-expansion-merging" pipeline (in Fig. 1) to maintain the integrity of the intermediate feature distribution of original transformers, preventing the loss of crucial learnable information during the accelerated training process. Similar to structure growth methods, we initially involve a limited number of tokens to participate in training and gradually grow the token number during training progress, eventually reaching the utilization of the entire token set. Then, a *widest feature-distribution token expansion* is introduced to make the feature distributions of the selected token set as wide as possible. Additionally, a *feature-distribution token merging* combines the tokens with close feature distributions to further avoid information loss. ToE not only accelerates the training and fine-tuning process of popular Transformers in a lossless manner or even with performance improvement, but also can be integrated into the existing efficient training frameworks (*e.g.*, Efficient-Train [30]) for further performance improvement, without twisting the original training hyper-parameters, architecture, and introducing additional training strategies.

Our main contributions can be summarized as follows:

- We propose ToE, a novel token growth scheme to accelerate ViTs from the perspective of tokens. ToE is a consistent training acceleration method and can be seamlessly integrated into the training and fine-tuning process of transformers without any modifications to the original training hyper-parameters, architecture, and strategies.
- We propose an effective "initialization-expansion-merging" framework to avoid the token information loss by maintaining the integrity of the intermediate feature distribution.
- Extensive experiments demonstrate that ToE accelerates the training and fine-tuning process of ViTs with a negligible accuracy drop or even surpassing the original full-token counterparts, which outperforms previous SOTA methods.

2. Related Work

2.1. Training Acceleration for Transformers

As mentioned above, many existing works focus on accelerating the training of transformers from the perspective of structural parameters. These structure methods [10, 12, 13, 27, 31, 32] reduce the number of updated parameters in the training process to save the computational cost. In contrast, the proposed ToE accelerates training from the perspective of reducing token redundancy. In other words, ToE computes a smaller number of tokens but still optimizes all parameters. It avoids potential performance drops in many structure growth methods due to the inconsistent structures of before-and-after models during structure growth and resetting of optimizer state when updating new structural parameters.

ToMe [11] uses a limited number of tokens to participate in training and progressively merges similar tokens layer-by-layer, which changes the attention operator in inference. ToE also involves merging tokens with close feature distributions by *feature-distribution token merging*. However, our merging strategy is performed only once at the end of the "initialization-expansion-merging" pipeline during training, which prevents the information loss of tokens. This ensures that ToE avoids the mismatch between practical and theoretical acceleration caused by excessive merging operations and operator modifications.

Additionally, several works [30, 33–35] also consider to reduce the data for training. The work in [33] deduplicates training datasets to save computational resources. Unfortunately, it usually introduces additional computational costs and sometimes becomes a bottleneck by using additional time to process datasets during training [36]. PSS [35] uses fewer patches obtained by splitting images during training. EfficientTrain [30] and PL [34] use images of different sizes and additional data augmentation. However, EfficientTrain and PL change the training pipelines that differ from the training of the original model, *e.g.*, hyper-parameters. Moreover, the above methods consider the properties of training data. In contrast, ToE focuses on the crucial learnable information in the intermediate feature space of transformers. Thus, ToE can be integrated into the above methods in a plug-and-play manner to further enhance training efficiency.

2.2. Training Acceleration for CNNs

Prior efficient training acceleration methods have explored ways to speed up the training of CNN models [37–42]. For example, works in [37, 38] consider pruning gradients to reduce training computation costs. Works in [39, 40] attempt to use quantization technical to achieve training acceleration. Others try to reduce training time either by reducing the number of optimization iterations with a linear decay for the learning rate [41] or skipping easy samples that contribute little to loss reduction [42]. However, these methods may not

be directly applied to Transformers for training acceleration due to the specific architectural differences between transformers and CNNs. Differently, ToE focuses on the training acceleration for Transformers on the token dimension.

2.3. Transformer pruning

Transformer pruning methods typically reduce parameters or tokens to generate sparse Transformers for fast inference. Structure pruning methods [14–17] attempted to prune the structures of transformers. Token pruning methods [18–22] focused on dynamically determining the importance of input tokens and pruning them during inference.

The key differences between our method and transformer pruning methods are two-fold. (1) Transformer pruning methods primarily aim to accelerate transformer inference, while our target is for training acceleration. (2) We obtain a dense model after training by token growth, which is entirely consistent with the original model for inference. In contrast, pruning methods generate sparse models after training.

3. Method

3.1. Preliminaries and Notations

Given a Transformer with L blocks, we denote the sets of input and output tokens for the l-th block as $\mathcal{S}_{l-1}$ and $\mathcal{S}_l$ with $l \in \{1, 2, \cdots, L\}$, respectively. The index set of output tokens for the l-th block is defined as $\mathcal{I} = \{1, 2, \cdots, N_l\}$, where N_l is the number of output tokens for the l-th block. We further denote the i-th token of the output tokens for the l-th block as $t_{l,i} \in \mathbb{R}^d$, thus $\mathcal{S}_l = \{t_{l,i} | \forall i \in \mathcal{I}\}$.

For the l-th Transformer block, we consider to reduce the output tokens to a specified size $N'_l = \lfloor rN_l \rfloor$, where $r \in (0, 1]$ is the kept rate of tokens, and $\lfloor \cdot \rfloor$ is a floor function. Further, we define the index set of kept tokens as $\mathcal{I}' = \{1, 2, \cdots, N'_l\}$ and we obtain a subset $\mathcal{S}'_l = \{t'_{l,i} | \forall i \in \mathcal{I}'\}$ of output tokens. When the output tokens of the l-th block are reduced, this results in a corresponding reduction in the quantity of input tokens for blocks beyond the l-th block. Furthermore, the computational complexity of self-attention blocks and MLP layers in Transformers is directly proportional to the number of input tokens. According to the work [43], the computation in the forward and backward propagation of modern neural networks roughly conforms to 1:2. Therefore, the reduction of tokens significantly accelerates the computation in both the forward and backward propagations during training if $r < 1$. Note that, to reduce the complex search computation for the kept rate of tokens r across all Transformer blocks, we simply and effectively set r to be the same in all blocks that benefit from acceleration.

3.2. Overview of ToE

As shown in Fig. 1, ToE initially selects a significantly small number of tokens, then progressively grows to the final full-

token same as the original Transformer, thereby achieving training acceleration. We divide the origin training process into N_g stages on average. We use a limited number of tokens to participate in each training stage and gradually grow the token number along with the training stages. The token growth strategy consists of three steps:

(1) Initial token selection as the seed tokens. we initially select $\lfloor r_0 N_l \rfloor$ output tokens from the origin token set $\mathcal{S}_l$ as the seed token set by using Uniform sampling on the index set $\mathcal{I}$, where r_0 represents the pre-defined initial kept rate, which is default set to less than 0.3 in our experiments unless otherwise specified.

(2) Token expansion. In the δ-th ($\delta \in \{1, 2, \cdots, N_g\}$) training stage, we perform δ times token expansion to preserve the integrity of the original intermediate feature space. Furthermore, we pre-define the keep rate of the first stage to be r_1. The kept rate of δ-th stage r_δ is computed as:

$$\mu_\delta = \begin{cases} r_1 - r_0, & \text{if } \delta = 1, \\ \frac{1-r_1}{N_g-1}, & \text{otherwise}, \end{cases} \tag{1}$$
$$r_\delta = r_{\delta-1} + \mu_\delta,$$

where μ_δ is the token expansion rate in the δ-th training stage and $r_1 = 2 \cdot r_0 \in (0, 1]$. After the δ times token expansion, we select $\lfloor r_\delta N_l \rfloor$ tokens from the full-token set $\mathcal{S}_l$. In Sec. 3.3.2, we will introduce the widest feature-distribution token expansion method to select $\lfloor r_\delta N_l \rfloor$ tokens, which aims to expand the token distribution space to effectively present full-token feature distribution.

(3) Token merging. To further avoid information loss during the training process, we consider merging the unselected tokens into the selected ones in the token expansion process, which retains effective information of the unselected tokens in the merged token set $\mathcal{S}'_l$. Inspired by ToMe [11], we merge *averagely* the tokens that the feature distributions are close as one new token, which is further introduced in Sec. 3.3.3.

During training, ToE performs steps (1), (2), and (3) on the original full-token set for each training iteration, which reduces the number of tokens involved in training while retaining the effective information from the full-token set.

3.3. Token Expansion

In this Section, we introduce the proposed ToE method, including spatial-distribution token initialization, widest feature-distribution token expansion, feature-distribution token merging, and its optimization.

3.3.1 Spatial-distribution Token Initialization

For the initialization, we apply a simple strategy to select the initial token set from $\mathcal{S}_l$. We define the index of the initial token set as:

$$\mathcal{I}^{(I)} = \{i | \forall i \bmod \lfloor \frac{1}{r_0} \rfloor = 1 \wedge \forall i \in \mathcal{I}\}. \tag{2}$$

The selected token set and the unselected tokens set can be expressed as $\mathbb{A} = \{t_{l,i} | \forall i \in \mathcal{I}^{(I)}\}$ and $\mathbb{B} = \mathcal{S}_l - \mathbb{A}$, respectively. This initialization selection strategy is based on spatial distribution. It indicates that we choose one token out of every $\lfloor \frac{1}{r_0} \rfloor$ tokens from the original token set and add it to the initial token set. Our strategy is simple, yet effective, to ensure that the initially selected tokens provide broad spatial coverage across the image patches.

3.3.2 Widest Feature-distribution Token Expansion

Previous works [11, 18] show that the intermediate feature space in modern Transformers is *overparameterized*, such that they prune the full-token Transformers to be sparse ones. Actually, through the above token initialization, we obtain the sparse Transformers. However, the performance drops significantly if we only train on these selected tokens. Thus, we consider to grow the number of tokens, which is expected to preserve the integrity of the original intermediate feature space and avoid the loss of tokens containing valuable information. Inspired by this, we seek to maintain the integrity of the intermediate feature distribution. Intuitively, when the feature distributions of two token sets are sufficiently close, they have similar information that can be used to effectively represent each other. In contrast, given one token whose feature distribution deviates significantly from all other tokens in the token set, it will be difficult to be adequately represented by other tokens, such that we expect to select this token to underscore its importance in the token expansion.

To this end, we propose the widest feature-distribution token expansion strategy. Specifically, we perform the expanding operation on the selected tokens from the initialized set. For the δ-th stage of token expansion, we consider the selected token set $\mathbb{A} \in \mathbb{R}^{|\mathbb{A}| \times d}$ and the unselected token set $\mathbb{B} \in \mathbb{R}^{|\mathbb{B}| \times d}$ as the 2D matrices, where $|\cdot|$ and d respectively denote the number of tokens and feature dimension, and $|\mathbb{A}| + |\mathbb{B}| = N_l$. We utilize *Cosine Distance* as the metric to measure the distance between feature distribution of tokens in these two sets (other metrics see Tab. 9):

$$\mathcal{D}(\mathbb{B}, \mathbb{A}) = \mathbf{1} - \cos \langle \mathbb{B}, \mathbb{A} \rangle = \mathbf{1} - \frac{\mathbb{B}\mathbb{A}^{\mathrm{T}}}{\|\mathbb{B}\| \cdot \|\mathbb{A}\|}, \tag{3}$$

where $\mathbf{1}$ is an all-one matrix. $\mathcal{D}(\mathbb{B}, \mathbb{A}) \in \mathbb{R}^{|\mathbb{B}| \times |\mathbb{A}|}$ represents the pairwise distances between tokens in $\mathbb{B}$ and $\mathbb{A}$.

We further define the distance between the feature distribution of tokens in $\mathbb{B}$ and its closest token in $\mathbb{A}$ as $distance(\mathbb{B} \to \mathbb{A}) \in \mathbb{R}^{|\mathbb{B}|}$:

$$distance(\mathbb{B} \to \mathbb{A})_i = \min_j(\mathcal{D}(\mathbb{B}, \mathbb{A})_{i,j}), \tag{4}$$

where $i \in \{1, \cdots, |B|\}$ and $j \in \{1, \cdots, |A|\}$. Eq. 4 indicates that we sample the minimal values of the feature-distribution distance matrix $\mathcal{D}(\mathbb{B}, \mathbb{A})$ along the second dimension. Thus, $distance(\mathbb{B} \to \mathbb{A})_i$ measures importance of
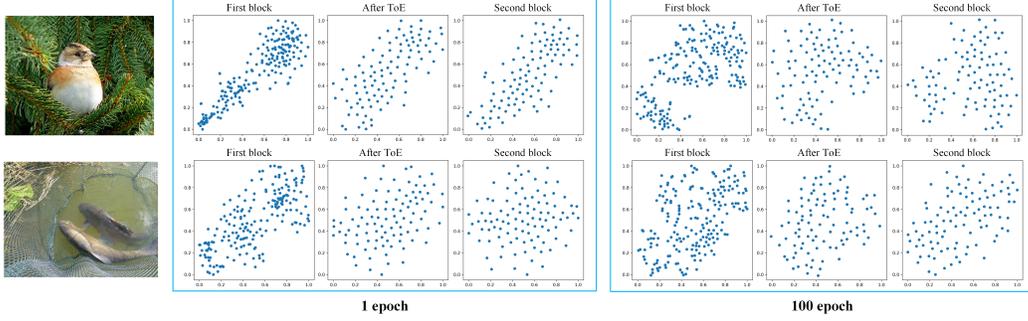
Figure 2. Visualization for the feature distribution of token set. We use T-SNE [44] to visualize the output token feature distributions at the first block, the tokens selected by ToE, and the output tokens after the second block. Baselines are DeiT-small trained on ImageNet-1K. ToE preserves the distribution integrity of intermediate features of the original token set across different Transformer blocks while ensuring that feature distributions are as wide as possible.

i-th token in $\mathbb{B}$. At this point, we progressively add the most important token to $\mathbb{A}$, which is formulated as:

$$\mathbb{A} = \mathbb{A} + t^*, \quad \mathbb{B} = \mathbb{B} - t^*,$$
$$t^* = \{\mathbb{B}_i | i = \arg\max(distance(\mathbb{B} \rightarrow \mathbb{A}))\}, \quad (5)$$

where t^* is the most important token in $\mathbb{B}$. When the feature distribution of one token is far from its closest token, it can be said that the feature distribution of this token deviates significantly from that of all other tokens in the token set. The operation described in Eq. 5 is performed for $\lfloor \mu_\delta N_l \rfloor$ times to select $\lfloor \mu_\delta N_l \rfloor$ tokens from $\mathbb{B}$ into $\mathbb{A}$. The widest feature-distribution token expansion strategy ensures that the feature distributions of the selected token set become as wide as possible, preventing the loss of important tokens. However, as we need to iterate $\lfloor \mu_\delta N_l \rfloor$ times expansion, it results in a considerable consumption of computational resources. Considering the computation parallelization, we modify the expanding operation in Eq. 5 parallelly:

$$\mathbb{A} = \mathbb{A} + \mathcal{S}^*, \quad \mathbb{B} = \mathbb{B} - \mathcal{S}^*,$$
$$\mathcal{S}^* = \{\mathbb{B}_i | i \in \text{topk}_{\lfloor \mu_\delta N_1 / k \rfloor}(distance(\mathbb{B} \rightarrow \mathbb{A}))\}, \quad (6)$$

where k is the pre-defined repetition step of parallel expanding operation, $\mathcal{S}^*$ is a token set consisting of the important tokens in $\mathbb{B}$, topk_n denotes the top argmax with the number of n tokens. By this way, we only perform k times parallel expanding operation to expand $\lfloor \mu_\delta N_l \rfloor$ tokens, and its computational consumption is negligible with small k.

3.3.3 Feature-distribution Token Merging

After token expansion, we aim to retain the effective information of the unselected tokens, such that we merge the unselected tokens that the feature distributions are close to the selected ones. The feature-distribution token merging can be formulated as:

$$\mathcal{S}'_l = \{\text{mean}(\mathbb{A}_j, \mathcal{S}_j^{(M)}) | \forall j \in \{1, 2, \cdots, |\mathbb{A}|\}\}, \text{where}$$
$$\mathcal{S}_j^{(M)} = \{\mathbb{B}_i | \mathcal{I}_i^{(M)} == j, \forall i \in \{1, 2, \cdots, |\mathbb{B}|\}\}, \quad (7)$$
$$\mathcal{I}^{(M)} = \arg\min_j(\mathcal{D}(\mathbb{B}, \mathbb{A})_{i,j}),$$

where $\mathcal{S}'_l \in \mathbb{R}^{|\mathbb{A}| \times d}$ is the token set merging the closest tokens from $\mathbb{B}$ to $\mathbb{A}$, and $\text{mean}(\mathbb{A}_j, \mathcal{S}_j^{(M)})$ indicate that we

merge $\mathbb{B}$ into $\mathbb{A}$ *averagely* based on the indice set $\mathcal{I}^{(M)} \in \mathbb{R}^{|\mathbb{B}|}$. Note that every $\mathbb{B}_i$ participates in the merging to avoid the information dropping for the unselected tokens.

3.3.4 Optimization of ToE

Our objective loss is the same as the original models, *e.g.*, cross-entropy loss in DeiT. The training details of ToE are presented in Algorithm 1. Note that we only apply ToE to the output tokens of the first transformer block. The detailed analysis is discussed in Sec. 4.4.

ToE is a plug-and-play acceleration module, which has three following advantages: (1) As shown in Fig. 2, we observed that the selected token set obtained by ToE in the multiple block outputs has a larger average distribution distance via T-SNE [44], compared to that in the original full-token set (see First block *vs.* After ToE). Moreover, it maintains a feature distribution similar to the original token set. It indicates ToE can preserve the integrity of the intermediate feature distribution of the original token set across different Transformer blocks by reducing the number of tokens. (2) ToE is a parameter-free module, it does not introduce any trainable parameters and utilizes efficient matrix calculations that the computational overhead is negligible, compared to computation-intensive self-attention. (3) The speedup factors (*e.g.*, token kept rate r_1 and training stage N_g) of ToE are independent of the original model's training hyper-parameters. This decoupling allows ToE to be seamlessly integrated into the training process of the original model, obviating the need for any adjustments to the training hyper-parameters.

4. Experiments

4.1. Experimental Settings

Datasets and baselines. We evaluate our method on ImageNet-1K [45] and CIFAR-10/100 [46]. For baselines, we use two popular ViTs, *i.e.*, DeiT [4] and LV-ViT [5], as the base models to evaluate the proposed ToE on ImageNet-1K. To further evaluate the universality, we integrate ToE into the efficient training framework EfficientTrain [30]. Moreover,

15787

Algorithm 1: Optimization with ToE

Input: Input dataset $\mathcal{X}$, output token number N_l, total training stage N_g, kept rate of the first training stage r_1, repetition step of the parallel expanding operation k, Transformer parameters θ, maximum iterations T.

Output: Updated Transformer parameters θ

1   **for** $t \leftarrow 1$ **to** T **do**
2   Sample from $\mathcal{X}$ to obtain data sample x, feed-forwarded through the embedding and first l-th transformer blocks to obtain the output token set $\mathcal{S}_l$;
3   *%%%Spatial-distribution Token Initialization%%%*
4      $r_0 \leftarrow \frac{1}{2}r_1$;
5   Initialize $\mathbb{A}$ and $\mathbb{B}$ by r_0, $\mathcal{S}_l$ via Eq. 2;
6   *%%%Widest Feature-distribution Token Expansion%%%*
7   Obtain the current training stage $\delta = \lceil N_g * t/T \rceil$;
8   **for** $m \leftarrow 1$ **to** δ **do**
9   **if** $m = 1$ **then**
10            $\mu_m \leftarrow r_1 - r_0$;
11   **else**
12            $\mu_m \leftarrow \frac{1-r_1}{N_g-1}$;
13   **end**
14   **for** $n \leftarrow 1$ **to** k **do**
15   Update $\mathbb{A}$ and $\mathbb{B}$ by μ_m, N_l, k, prior $\mathbb{A}$ and prior $\mathbb{B}$ via Eq. 6;
16   **end**
17   **end**
18   *%%%Feature-distribution Token Merging%%%*
19   Obtain $\mathcal{S}_l'$ by $\mathbb{A}$ and $\mathbb{B}$ via Eq. 7;
20      $\mathcal{S}_l'$ feed-forwarded through the l+1-th transformer block to final layer and progressively obtain the final prediction y;
21   *%%%Parameter Updating%%%*
22   Use y to compute the loss and obtain the gradient $\nabla\theta$;
23   Use $\nabla\theta$ to update prior θ via the optimizer to obtain new θ;
24   **end**
25   **return** θ

---

we evaluate the transfer learning ability using pre-trained weights of ToE on DeiT and the performance of accelerating the fine-tuning process with ToE on CIFAR-10/100.

Evaluation metrics. We report Top-1 accuracy, the GPU training time and FLOPs as the evaluation metric. To evaluate the training speed, we report the total GPU hours consumed during the entire training process, as well as the theoretical FLOPs for *one forward-backward process*. To avoid the impact of memory access and kernel launching on training time [12], we report the GPU hours on different numbers of GPUs, but with the same GPU numbers to evaluate different training methods. The FLOPs for the forward process are measured using thop [1], and for the backward process, we follow [43] and calculate it as twice the FLOPs of the forward process.

Implementations. All methods are trained by Pytorch [47]. For DeiT and LV-ViT, all experiments are conducted on four NVIDIA RTX A6000 GPUs[2], while EfficientTrain is trained on eight NVIDIA RTX A6000 GPUs.

All hyper-parameters (*e.g.*, learning rate, decay strategy and rate), and training strategies and optimization processes are the same as the original papers unless otherwise specified.

Growth strategy. In default, we divide the origin training process into $N_g = 3$ stages on average. The token kept rate of 1st stage r_1 is set to 0.4, 0.5 or 0.6, our method is corresponding to be denoted as ToE $_{r_1=0.4}$, ToE $_{r_1=0.5}$ or ToE $_{r_1=0.6}$. Correspondingly, the kept rate of the initial stage r_0 is set to 0.2, 0.25 and 0.3. The repetition step of parallel expanding operation k is default set to 2, and we perform ToE on the output tokens of the first block for all models.

4.2. Results on ImageNet-1k

DeiT and LV-ViT As shown in Tab. 2, ToE achieves lossless training acceleration with SOTA performance. For example, ToE $_{r_1=0.5}$ achieves 0.4% Top-1 accuracy improvement with $1.27\times$ theoretical and $1.24\times$ practical faster speed to train DeiT-tiny. For DeiT-small, it achieves $1.3\times$ training acceleration without accuracy drop. Compared to the SOTA methods, ToE $_{r_1=0.5}$ outperforms SViTE [10] and NetworkExpansion [12] at least 1% Top-1 accuracy at the consistent acceleration ratio for training both DeiT-tiny and DeiT-small. Compared to ToMe [11], ToE $_{r_1=0.5}$ also achieves both higher accuracy and practical training speed. Note that ToMe is able to reduce GFLOPs, but fails to accelerate training due to the usage of unfriendly weighted average attention and layer-wise merging operations. For DeiT-base, ToE $_{r_1=0.5}$ drops only 0.2% Top-1 accuracy while saving more than 60 GPU hours in the practical training process, which is comparable to NetworkExpansion with EMA. If we relax the restriction of hyper-parameter consistency (presented in Appendix), ToE $_{r_1=0.4}^{\text{Hyper}}$ outperforms NetworkExpansion with 0.2% accuracy and 24h training time reduction.

For LV-ViT-T and LV-ViT-S shown in Tab. 3, ToE $_{r_1=0.4}$ achieves efficient training with $1.2\times$ acceleration rate, while without accuracy drop or even with accuracy improvement for training LV-ViT-T, compared to baselines. Note that the results of ToE $_{r_1=0.4}$ and NetworkExpansion are reported with EMA, due to the default LV-ViT training with EMA. In addition, ToE $_{r_1=0.4}$ outperforms NetworkExpansion in both training acceleration and accuracy with 0.5h training time reduction and 0.6% accuracy for LV-ViT-T, respectively.

We also present the validation Top-1 accuracy of ToE and NetworkExpansion during training DeiT-tiny and LV-ViT-T in Fig. 3. As observed, ToE initially reduces token redundancy during training, resulting in some performance drops compared to the baseline. However, in the later stages of training, ToE introduces more tokens for training, gradually reducing the accuracy gap to the baseline. Benefiting from the reduction of token redundancy in the early stages, models trained by ToE with the proposed token expansion and merging achieve higher accuracies, compared to baselines. Compared to NetworkExpansion, our ToE is more stable to

Table 2. Performance comparison for DeiT on ImageNet-1K. a/b in the column of Top-1 Acc. means without/with EMA strategy using the official GitHub repo[†]. The training time is averagely measured on one/two/four NVIDIA RTX A6000 GPUs for DeiT-tiny/small/base 3 times, and the batch size is set to 1,024 in all following tables and figures.

Model	Method	Consistency			Top-1 Acc. (%)	GFLOPs (per training iter)	Training time (total GPU hours)	Acceleration (practical rate)
		Hyper?	Architecture?	Strategy?				
DeiT-tiny	Baseline [4]	–	–	–	72.2	3.3×10^3	54.6h	1.00×
	(NeurIPS'21) S^{2}ViTE-Tiny (600 epoch) [10]	×	×	✓	70.1 (-2.1)	2.5×10^3 (1.32×)	–	1.19×
	(ICLR'23) ToMe$_{r_s \to}^{DeiT}$ [11]	✓	×	✓	71.7 (-0.5)	2.5×10^3 (1.32×)	53.3h	1.02×
	(CVPR'23) NetworkExpansion$_{6 \to 12}$ [12]	✓	✓	×	70.3 (-1.9) / 70.1 (-2.1)	2.5×10^3 (1.32×)	43.2h	1.26×
	ToE $_{r_1=0.5}$ (Ours)	✓	✓	✓	**72.6 (+0.4)**	2.6×10^3 (1.27×)	44.2h	1.24×
DeiT-small	Baseline [4]	–	–	–	79.8	1.3×10^4	124.5h	1.00×
	(ICLR'23) ToMe$_{r_s \to}^{DeiT}$ [11]	✓	×	✓	79.7 (-0.1)	9.8×10^3 (1.33×)	121.5h	1.02×
	(CVPR'23) NetworkExpansion$_{6 \to 12}$ [12]	✓	✓	×	78.8 (-1.0) / 78.6 (-1.2)	9.8×10^3 (1.33×)	100.3h	1.24×
	ToE $_{r_1=0.5}$ (Ours)	✓	✓	✓	**79.8 (+0.0)**	1.0×10^4 (1.30×)	102.2h	1.22×
DeiT-base	Baseline [4]	–	–	–	81.8	5.2×10^4	292.8h	1.00×
	(ICML'19) StackBERT [13]	✓	✓	×	80.8 (-1.0)	4.2×10^4 (1.24×)	231.6h	1.26×
	(CVPR'23) NetworkExpansion$_{6 \to 12}$ [12]	✓	✓	×	81.0 (-0.8) / 81.5 (-0.3)	3.9×10^4 (1.33×)	226.8h	1.29×
	ToE $_{r_1=0.5}$ (Ours)	✓	✓	✓	81.6 (-0.2)	4.0×10^4 (1.30×)	231.2h	1.27×
	ToE $_{r_1=0.4}$ (Ours)	✓	✓	✓	81.4 (-0.4)	3.8×10^4 (1.37×)	225.2h	1.30×
	ToE $_{r_1=0.5}^{Hyper}$ (Ours)	×	✓	✓	**81.8 (+0.0)**	3.6×10^4 (1.44×)	213.2h	1.37×
	ToE $_{r_1=0.4}^{Hyper}$ (Ours)	×	✓	✓	81.7 (-0.1)	3.3×10^4 (1.58×)	202.8h	1.44×

[†] https://github.com/huawei-noah/Efficient-Computing/tree/master/TrainingAcceleration/NetworkExpansion

Table 3. Performance comparison for LV-ViT on ImageNet-1K. ‡ indicates that results reproduced by the official GitHub repo. The training time is averagely measured on two/four NVIDIA RTX A6000 GPUs 3 times for LV-ViT-T/S with a fixed batch size of 1,024.

Model	Method	Top-1 Acc. (%)	GFLOPs (per training iter)	Training time (total GPU hours)
LV-ViT-T	Baseline [5]	79.1	8.2×10^3	130.5h
	(CVPR'23) NetworkExpansion$_{6 \to 12}$ [12]	‡78.8 (-0.3)	7.1×10^3 (1.14×)	114.4h (1.14×)
	ToE$_{r_1=0.4}$ (Ours)	**79.4 (+0.3)**	7.0×10^3 (1.17×)	113.9h (1.15×)
LV-ViT-S	Baseline [5]	83.3	1.9×10^4	237.3h
	(CVPR'23) NetworkExpansion$_{8 \to 16}$ [12]	‡82.9 (-0.4)	1.5×10^4 (1.27×)	195.5h (1.21×)
	ToE$_{r_1=0.4}$ (Ours)	**83.3 (+0.0)**	1.4×10^4 (1.36×)	195.3h (1.22×)
LV-ViT-M	Baseline [5]	84.1	3.7×10^4	368.7h
	(CVPR'23) NetworkExpansion$_{10 \to 20}$ [12]	84.0 (-0.1)	2.9×10^4 (1.28×)	292.7h (1.26×)
	ToE $_{r_1=0.4}$ (Ours)	**84.1 (+0.0)**	2.7×10^4 (1.37×)	292.5h (1.26×)

Table 4. Performance comparison between EfficientTrain [30] and our combination framework on ImageNet-1K.

Model	Method	Top-1 Acc. (%)	GFLOPs (per training iter)	Training time (total GPU hours)
DeiT-tiny	Baseline (EfficientTrain) [30]	72.5	1.3×10^4	52.5h
	(ICCV'23) EfficientTrain [30]	73.3 (+0.8)	8.8×10^3 (1.48×)	36.5h (1.44×)
	EfficientTrain + ToE $_{r_1=0.6}$ (Ours)	**73.5 (+1.0)**	7.6×10^3 (1.71×)	32.3h (1.63×)
DeiT-small	Baseline (EfficientTrain) [30]	80.3	5.2×10^4	121.3h
	(ICCV'23) EfficientTrain [30]	**80.4 (+0.1)**	3.4×10^4 (1.53×)	85.2h (1.42×)
	EfficientTrain + ToE $_{r_1=0.6}$ (Ours)	**80.4 (+0.1)**	2.9×10^4 (1.79×)	79.4h (1.53×)

train with consistent accuracy improvement during training, while the accuracy of NetworkExpansion with EMA drops significantly at the intermediate epoch number and then restores due to the inconsistent structures of before-and-after models when structure growing. More validation curves are presented in the Appendix.

Combination with EfficientTrain [30]. ToE can be seamlessly integrated into the EfficientTrain framework to further improve the performance. We do not modify the pipeline of EfficientTrain and simply apply ToE to the output tokens of the model's first block. The results are summarized in Tab. 4, which effectively evaluates the universality of ToE. The combination of EfficientTrain and ToE achieves higher training speeds to further enhance the training efficiency of EfficientTrain with accuracy improvement.

4.3. Transfer Results on CIFAR-10/100

we further explore the transfer learning ability of ToE-pretrained weights and evaluate whether ToE can be used to accelerate the fine-tuning on CIFAR-10/100. For the fine-
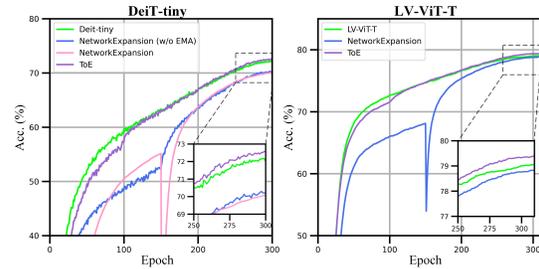

Figure 3. Validation Top-1 accuracy of DeiT-tiny and LV-ViT-T on ImageNet-1k during training with different methods.

Table 5. Results for fine-tuning DeiT on CIFAR-10/100.

Model	Pre-training		Fine-tuning		Top-1 Acc. (%)	
	Method	Acceleration	Method	Acceleration	CIFAR-10	CIFAR-100
DeiT-tiny	Baseline [4]	1.0x	Baseline [4]	1.0x	98.07	86.78
	Baseline [4]	1.0x	ToE$_{r_1=0.5}$	1.3x	98.10 (+0.03)	86.74 (-0.04)
	ToE$_{r_1=0.5}$	1.3x	Baseline [4]	1.0x	**98.19 (+0.12)**	**87.10 (+0.32)**
	ToE$_{r_1=0.5}$	1.3x	ToE$_{r_1=0.5}$	1.3x	98.16 (+0.09)	86.91 (+0.13)
DeiT-small	Baseline [4]	1.0x	Baseline [4]	1.0x	98.93	90.15
	Baseline [4]	1.3x	ToE$_{r_1=0.5}$	1.3x	98.96 (+0.03)	90.19 (+0.04)
	ToE$_{r_1=0.5}$	1.3x	Baseline [4]	1.0x	**99.03 (+0.10)**	**90.37 (+0.22)**
	ToE$_{r_1=0.5}$	1.3x	ToE$_{r_1=0.5}$	1.3x	98.99 (+0.06)	90.26 (+0.11)

tuning settings, we follow the settings of the official GitHub repo [3]. We introduce the training details in the Appendix.

As shown in Tab. 5, pre-training weights by ToE is able to improve the accuracy on CIFAR-10/100 for DeiT-tiny/small, when using the same baseline training for fine-tuning (see the 1st and 3rd rows in both DeiT-tiny and DeiT-small). For example, ToE pre-training outperforms baseline pre-training by 0.32% accuracy on CIFAR-100, which evaluates the strong transfer ability of ToE. In addition, our ToE is also effective and efficient for fine-tuning (see the 1st and 2nd rows in DeiT-tiny/small). ToE achieves 1.3× acceleration for fine-tuning DeiT-tiny with 0.03 accuracy improvement on CIFAR-10. Further, we employ ToE for both pre-training and fine-tuning, which significantly accelerates the training with an accuracy improvement of at least 0.06% on CIFAR-10 for both DeiT-tiny/small, compared to that using both baselines.

---

[3]https://github.com/facebookresearch/deit

Table 6. Ablation studies of different speedup factors for DeiT-tiny on ImageNet-1K. The default r_0/r_1, N_g and k are set to $1/2$, 3 and 2, respectively. All results in this table have almost the same training speeds for 44h training (total GPU hours).

	Factors	$r_0/r_1 = 1/3$	$r_0/r_1 = 2/3$	$N_g = 2$	$N_g = 4$	$k = 1$	$k = 3$	default
DeiT-tiny	Top-1 Acc. (%)	72.3	72.5	72.4	72.5	72.5	72.6	72.6

Table 7. Effect of "initialization-expansion-merge" pipeline for DeiT on ImageNet-1K. $\pm$ indicates we conduct 3 runs to calculate the mean and std.

Initialization		Expansion	Merge	Top-1 Acc. (%)	
Random	Spatial			DeiT-tiny	DeiT-small
$\times$	$\checkmark$	$\checkmark$	$\checkmark$	**72.6**	**79.8**
$\checkmark$	$\times$	$\checkmark$	$\checkmark$	72.3±0.2	79.7±0.1
$\times$	$\checkmark$	$\times$	$\checkmark$	71.2	79.1
$\times$	$\checkmark$	$\checkmark$	$\times$	71.7	79.6

Table 8. Results of applying ToE to different early transformer block's output tokens for DeiT-tiny on ImageNet-1K.

Block	Top-1 Acc. (%) DeiT-tiny	GFLOPs (per training iter)	Training time (total GPU hours)
Embedding	72.1	2.51×10^3	43.5h
First block	**72.6**	2.58×10^3	44.2h
Second block	72.2	2.65×10^3	45.2h
Third block	72.1	2.71×10^3	46.9h

4.4. Ablation Study

Effect of speedup factors in ToE. As presented in Tab. 6, we verify the sensitivity of the speedup factors mentioned in Sec. 3.3, such as the ratio of r_0/r_1, training stages N_g and parallel expanding operation k. At almost the same training time, ToE is relatively insensitive to these factors, *w.r.t* accuracy. It allows ToE to be easily integrated into the different models' training pipeline with minimal factor adjustments.

We further adjust the keep rate of the first stage r_1 to control the training speed, and the relationship between r_1 and training speed is illustrated in Fig. 4. We found ToE achieves more than $1.3\times$ acceleration on DeiT-tiny without accuracy dropping. Additionally, it also demonstrates that reducing token redundancy in the early stages of training sometimes improves the model performance.

Effect of "Initialization-expansion-merging". Tab. 7 provides an analysis of the necessity of each step in the proposed "initialization-expansion-merging" pipeline. When we randomly select tokens as the initial token set rather than *spatial-distribution token initialization*, it leads to the performance degradation. Furthermore, removing *widest feature-distribution token expansion* and *feature-distribution token merging* from the pipeline significantly decreases the accuracy, *e.g.*, more than 0.9% and 1.4% accuracy drops without the merging and expansion for DeiT-tiny, respectively.

Where to apply ToE. Work in [32, 48] demonstrates that class attention tends to be a global pooling as more attention operations are performed, and tokens in early blocks are more similar. This leads to more redundancy in tokens from early blocks. Consequently, applying ToE to the output tokens of early blocks can achieve higher acceleration. As shown in Tab. 8, we default apply ToE to the output tokens of the first block, which achieves the best trade-off between accuracy and training speed, compared to other early blocks.
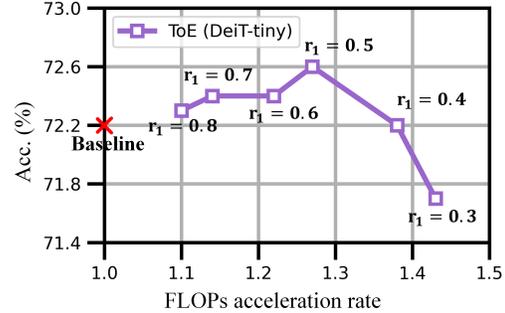

Figure 4. Trade-off between acceleration ratio and model performance by setting different r_1.

Table 9. Results of different feature-distribution distances in Eq. 3 for DeiT on ImageNet-1K.

Measure	Top-1 Acc. (%)	
	DeiT-tiny	DeiT-small
Manhattan Distance	69.8	78.0
Euclidean Distance	70.6	78.4
Cosine Distance	**72.6**	**79.8**

Effect of the feature-distribution distance. We explore the metric that measures the feature-distribution distance between two tokens in Eq. 3. As shown in Tab. 9, we use three different metrics: Manhattan distance, Euclidean distance, and Cosine distance. We observe that Cosine distance achieves the best performance as the distance metric.

5. Conclusion

In this paper, we proposed a novel token growth scheme Token Expansion (ToE) to achieve consistent training acceleration for ViTs. ToE introduce an "initialization-expansion-merging" pipeline to maintain the integrity of the intermediate feature distribution of original transformers, preventing the loss of crucial learnable information in the training process. In experiments, ToE can be seamlessly integrated into the training of various transformers and efficient training frameworks in a lossless manner or even accuracy improvement, compared to the entire full-token training. These experimental results of ToE also demonstrate the superior performance gains over the SOTA methods.

Acknowledgements

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 30, 2017. 1

[2] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, 2019.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 33:1877–1901, 2020. 1

[4] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICLR*, pages 10347–10357. PMLR, 2021. 1, 2, 5, 7

[5] Zi-Hang Jiang, Qibin Hou, Li Yuan, Daquan Zhou, Yujun Shi, Xiaojie Jin, Anran Wang, and Jiashi Feng. All tokens matter: Token labeling for training better vision transformers. *NeurIPS*, 34:18590–18602, 2021. 5, 7

[6] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, pages 213–229. Springer, 2020.

[7] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *NeurIPS*, 34:12077–12090, 2021. 1

[8] 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. In *ICLR*, 2020. 1

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1

[10] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. *NeurIPS*, 34:19974–19988, 2021. 1, 2, 3, 6, 7

[11] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. In *ICLR*, 2022. 1, 2, 3, 4, 6, 7

[12] Ning Ding, Yehui Tang, Kai Han, Chao Xu, and Yunhe Wang. Network expansion for practical training acceleration. In *CVPR*, pages 20269–20279, 2023. 1, 2, 3, 6, 7

[13] Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of bert by progressively stacking. In *ICML*, pages 2337–2346. PMLR, 2019. 1, 2, 3, 7

[14] Huanrui Yang, Hongxu Yin, Maying Shen, Pavlo Molchanov, Hai Li, and Jan Kautz. Global vision transformer pruning with hessian-aware saliency. In *CVPR*, pages 18547–18557, 2023. 1, 3

[15] Fang Yu, Kun Huang, Meng Wang, Yuan Cheng, Wei Chu, and Li Cui. Width & depth pruning for vision transformers. In *AAAI*, volume 36, pages 3143–3151, 2022.

[16] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. Block pruning for faster transformers. In *EMNLP*, pages 10619–10629, 2021.

[17] Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate models. In *ACL*, pages 1513–1528, 2022. 1, 3

[18] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *NeurIPS*, 34:13937–13949, 2021. 1, 3, 4

[19] Lingchen Meng, Hengduo Li, Bor-Chun Chen, Shiyi Lan, Zuxuan Wu, Yu-Gang Jiang, and Ser-Nam Lim. Adavit: Adaptive vision transformers for efficient image recognition. In *CVPR*, pages 12309–12318, 2022.

[20] Mohsen Fayyaz, Soroush Abbasi Koohpayegani, Farnoush Rezaei Jafari, Sunando Sengupta, Hamid Reza Vaezi Joze, Eric Sommerlade, Hamed Pirsiavash, and Jürgen Gall. Adaptive token sampling for efficient vision transformers. In *ECCV*, pages 396–414. Springer, 2022.

[21] Zhenglun Kong, Peiyan Dong, Xiaolong Ma, Xin Meng, Wei Niu, Mengshu Sun, Xuan Shen, Geng Yuan, Bin Ren, Hao Tang, et al. Spvit: Enabling faster vision transformers via latency-aware soft token pruning. In *ECCV*, pages 620–640. Springer, 2022.

[22] Hongxu Yin, Arash Vahdat, Jose M Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-vit: Adaptive tokens for efficient vision transformer. In *CVPR*, pages 10809–10818, 2022. 1, 3

[23] Sheng Xu, Yanjing Li, Mingbao Lin, Peng Gao, Guodong Guo, Jinhu Lü, and Baochang Zhang. Q-detr: An efficient low-bit quantized detection transformer. In *CVPR*, pages 3842–3851, 2023. 1

[24] Yanjing Li, Sheng Xu, Baochang Zhang, Xianbin Cao, Peng Gao, and Guodong Guo. Q-vit: Accurate and fully quantized low-bit vision transformer. *NeurIPS*, 35:34451–34463, 2022.

[25] Yefei He, Zhenyu Lou, Luoming Zhang, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. Bivit: Extremely compressed binary vision transformers. In *ICCV*, pages 5651–5663, 2023.

[26] Phuoc-Hoan Charles Le and Xinlin Li. Binaryvit: Pushing binary vision transformers towards convolutional models. In *CVPR*, pages 4664–4673, 2023. 1

[27] Cheng Chen, Yichun Yin, Lifeng Shang, Xin Jiang, Yujia Qin, Fengyu Wang, Zhi Wang, Xiao Chen, Zhiyuan Liu, and Qun Liu. bert2bert: Towards reusable pretrained language models. In *ACL*, pages 2134–2148, 2022. 1, 2, 3

[28] Xin Yuan, Pedro Savarese, and Michael Maire. Growing efficient deep networks by structured continuous sparsification. In *ICLR*, 2021.

[29] Wei Wen, Feng Yan, Yiran Chen, and Hai Li. Autogrow: Automatic layer growing in deep convolutional networks. In *KDD*, pages 833–841, 2020. 1

[30] Yulin Wang, Yang Yue, Rui Lu, Tianjiao Liu, Zhao Zhong, Shiji Song, and Gao Huang. Efficienttrain: Exploring generalized curriculum learning for training visual backbones. In *ICCV*, pages 5852–5864, 2023. 2, 3, 5, 7

[31] Changlin Li, Bohan Zhuang, Guangrun Wang, Xiaodan Liang, Xiaojun Chang, and Yi Yang. Automated progressive learning for efficient training of vision transformers. In *CVPR*, pages 12486–12496, 2022. 3

[32] Xuran Pan, Xuan Jin, Yuan He, Shiji Song, Gao Huang, et al. Budgeted training for vision transformer. In *ICLR*, 2022. 3, 8

[33] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. In *ACL*, pages 8424–8445, 2022. 3

[34] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *ICLR*, pages 10096–10106. PMLR, 2021. 3

[35] Bradley McDanel and Chi Phuong Huynh. Accelerating vision transformer training via a patch sampling schedule. *arXiv preprint arXiv:2208.09520*, 2022. 3

[36] Li Shen, Yan Sun, Zhiyuan Yu, Liang Ding, Xinmei Tian, and Dacheng Tao. On efficient training of large-scale deep learning models: A literature review. *arXiv preprint arXiv:2304.03589*, 2023. 3

[37] Yuedong Yang, Guihong Li, and Radu Marculescu. Efficient on-device training via gradient filtering. In *CVPR*, pages 3811–3820, 2023. 3

[38] Xucheng Ye, Pengcheng Dai, Junyu Luo, Xin Guo, Yingjie Qi, Jianlei Yang, and Yiran Chen. Accelerating cnn training by pruning activation gradients. In *ECCV*, pages 322–338. Springer, 2020. 3

[39] Yonggan Fu, Haoran You, Yang Zhao, Yue Wang, Chaojian Li, Kailash Gopalakrishnan, Zhangyang Wang, and Yingyan Lin. Fractrain: Fractionally squeezing bit savings both temporally and spatially for efficient dnn training. *NeurIPS*, 33:12127–12139, 2020. 3

[40] Yue Wang, Ziyu Jiang, Xiaohan Chen, Pengfei Xu, Yang Zhao, Yingyan Lin, and Zhangyang Wang. E2-train: Training state-of-the-art cnns with over 80% energy savings. *NeurIPS*, 32, 2019. 3

[41] Mengtian Li, Ersin Yumer, and Deva Ramanan. Budgeted training: Rethinking deep neural network training under resource constraints. In *ICLR*, 2019. 3

[42] Jiong Zhang, Hsiang-Fu Yu, and Inderjit S Dhillon. Autoassist: A framework to accelerate training of deep neural networks. *NeurIPS*, 32, 2019. 3

[43] Marius Hobbhahn and Jaime Sevilla. What's the backward-forward flop ratio for neural networks? https://epochai.org/blog/backward-forward-FLOP-ratio, 2021. Accessed: 2023-9-28. 3, 6

[44] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 9:2579–2605, 2008. 5

[45] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. Ieee, 2009. 5

[46] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009. 5

[47] 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*, 32, 2019. 6

[48] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *NeurIPS*, 34:12116–12128, 2021. 8