Network Expansion For Practical Training Acceleration

Ning Ding^{1,2}, Yehui Tang^{1,2}, Kai Han^{2*}, Chao Xu^{1}, Yunhe Wang^{2*}
^{1} National Key Lab of General AI, School of Intelligence Science and Technology, Peking University
^{2} Huawei Noah's Ark Lab
dingning@stu.pku.edu.cn, yhtang@pku.edu.cn, kai.han@huawei.com,
xuchao@cis.pku.edu.cn, yunhe.wang@huawei.com

Abstract

*Recently, the sizes of deep neural networks and training datasets both increase drastically to pursue better performance in a practical sense. With the prevalence of transformer-based models in vision tasks, even more pressure is laid on the GPU platforms to train these heavy models, which consumes a large amount of time and computing resources as well. Therefore, it's crucial to accelerate the training process of deep neural networks. In this paper, we propose a general network expansion method to reduce the practical time cost of the model training process. Specifically, we utilize both width- and depth-level sparsity of dense models to accelerate the training of deep neural networks. Firstly, we pick a sparse sub-network from the original dense model by reducing the number of parameters as the starting point of training. Then the sparse architecture will gradually expand during the training procedure and finally grow into a dense one. We design different expanding strategies to grow CNNs and ViTs respectively, due to the great heterogeneity in between the two architectures. Our method can be easily integrated into popular deep learning frameworks, which saves considerable training time and hardware resources. Extensive experiments show that our acceleration method can significantly speed up the training process of modern vision models on general GPU devices with negligible performance drop (*e.g. *1.42× faster for ResNet-101 and 1.34× faster for DeiT-base on ImageNet-1k). The code is available at https://github.com/huawei-noah/Efficient-Computing/tree/master/TrainingAcceleration/NetworkExpansion and https://gitee.com/mindspore/hub/blob/master/mshub_res/assets/noah-cvlab/gpu/1.8/networkexpansion_v1.0_imagenet2012.md.*

1. Introduction

Deep neural networks have demonstrate their excellent performance on multiple vision tasks, such as classification [15, 30, 44], object detection [12, 43], semantic segmentation [32, 35], *etc.* In spite of their success, these networks usually come with heavy architectures and severe over-parameterization, and therefore it takes many days or even weeks to train such networks from scratch. The ever-increasing model complexity [23, 24, 34, 42] and training time cause not only a serious slowdown for the research schedule, but also a huge waste of time and computing resources. However, CNNs are still going deeper and bigger for higher capacity to cope with extremely large datasets [27, 45]. Recently, a new type of architecture named vision transformers (ViTs) have emerged and soon achieved state-of-the-art performances on multiple computer vision tasks [16, 48, 52, 57]. Originating from Natural Language Processing, the vision transformer has a different network topology and larger computational complexity than CNNs. Besides, transformer-based models usually require more epochs to converge.

From another perspective, compared with purchasing expensive GPU servers, many researchers and personal users nowadays choose cloud computing service to run experiments and pay their bills by GPU-hours. Thus, an accelerated training framework is obviously cost-efficient. On the other hand, shortened training time leads to not only quicker idea verification but also more refined hyper-parameter tuning, which is crucial to the punctual completion of the project and on-time product delivery.

There are some existing methods about efficient model training [36, 51, 53, 55], but few of them can achieve high practical acceleration on geneal GPU platforms. [53] proposes to prune the gradients of feature maps during back-propagation to reduce train-time FLOPs, and achieve training speedup on CPU platform. [51] conducts efficient CNN training on ARM and FPGA devices to reduce power consumption. [36] prunes weights of the network to achieve training acceleration but eventually yield a pruned sparse model with non-negligible performance drop. [55] skips easy samples that contribute little to loss reduction by using an assistant model asynchronously running on CPU. Yet it

⋆ Corresponding authors.

requires sophisticated engineering implementation. Though the prior works claim an ideal theoretical acceleration ratio, none of them achieve obviously practical acceleration on common GPU platforms. Most of these works overlook the most general scenario, *i.e*. accelerating training on general GPU platforms with popular deep learning frameworks such as PyTorch [40] and TensorFlow [1]. The lack of related research is probably because GPU servers are not so power-constrained as edge devices.

In this paper, we propose a general training acceleration framework (network expansion) for both CNN and ViT models to reduce the practical training time. We first sample a sub-network from the original dense model as the starting point of training. Then this sparse architecture will gradually expand its network topology by adding new parameters, which increases the model capacity along the training procedure. When performing network expansion, we follow the principle of avoiding the introduction of redundant parameters. For CNN, new filters are progressively added whose weights are initialized by imposing filter-level orthogonality. This reduces the correlation between old and new feature maps and improves the expressiveness of the convolutional network. For vision transformers, we first train a shallow sub-network with fewer layers, and create an exponential moving average (EMA) version of the trained model. As the training continues, some layers of the EMA model will be inserted into the trained model to construct a deeper one. With the network expansion training paradigm, the sampled sub-network eventually grows into the desired dense architecture, and thus the total training FLOPs and time are greatly reduced.

Our method can be easily integrated into popular deep learning frameworks on general GPU platforms. Without changing the original optimizer and hyper-parameters (such as epochs and learning rate), our method can achieve 1.42× wall-time acceleration for training ResNet-101, 1.34× wall-time acceleration for training DeiT-base, on ImageNet-1k dataset with negligible top-1 accuracy gap, compared with normal training baseline. Moreover, experiments show that our acceleration framework can generalize to downstream tasks such as semantic segmentation.

2. Related Work

There are lots of works that study the accelerated training methods for deep neural networks in many respects, since the lengthy training phase is a practical issue to be tackled for real-world applications of deep learning algorithms.

2.1. Efficient Training on Edge Devices

Net2Net [6] proposes to reuse the weights of a pre-trained small model to initialize a large-sized model, which results in faster convergence of the new model. [53] proposes to prune the gradients of feature maps during back-propagation stage to reduce train-time FLOPs. However this method needs to modify runtime library and achieves training speedup only on CPU platform. E2-train [51] conducts efficient CNN training on edge devices, such as ARM and FPGA, to reduce both the time under the constraint of limited power. PruneTrain [36] simultaneously prunes weights of the network and train the network to reduce FLOPs, and thus eventually yielding a pruned sparse model rather than a complete dense model. [56] also proposes to speed up training by trying out different possible sub-networks inside the dense network and obtain a sparse model. AutoAssist [55] identifies and skips easy samples that contribute little to the loss reduction by using an assistant model asynchronously running on CPU. Yet it requires sophisticated engineering implementation.

2.2. Distributed Parallel Training

There is another scope of research regarding accelerated training. These works [2, 14, 26, 54] try to train neural networks on large-scale dataset, such as ImageNet, within just a few minutes using distributed parallel computing cluster. [54] efficiently trains ResNet50 on ImageNet within 20 minutes by using 2,048 high-end Intel CPU. [2] uses 1,024 Tesla P100 GPUs to complete the training of ResNet50 on ImageNet with the batch size being 32K. In particular, [26] utilizes a cluster system of 2,048 Tesla P40 GPUs to train AlexNet within 4 minutes and ResNet50 in 6.6 minutes. The hardware scale adopted by these works can only be afforded by leading enterprises, and is never possible for normal researchers and cloud-service users. Thus they are out of the research scope of this paper.

2.3. Efficient Training of Language Model

There are a few researches that work on the efficient training of pretrained language models (PLM), such as BERT. Bert2BERT [5] reuses the parameters of a thinner BERT model to initialize a wide BERT while maintaining the mapping function, in order to accelerate the training process. StackBERT [13] copies the whole transformer layers of a pretrained shallow model and directly stack them on the top of it to build a deep model, claiming that this will transfer the well-learned knowledge of the shallow model into the deep one, thus speeding up the training. However, these methods are sub-optimal for vision tasks and will introduce performance drop when applied to vision transformers which take images as input.

2.4. Network pruning

Network pruning algorithms aim to find a sparse network from a pretrained dense one with marginal performance drop in pursuit of faster inference speed. However, pruning algorithms would instead take much longer time to train networks than conventional training procedures because of necessary finetuning [17, 18, 21, 33, 46, 47]. Therefore, some recent works [7, 22, 38] called one-shot pruning try to finish the pruning process by training only once without any

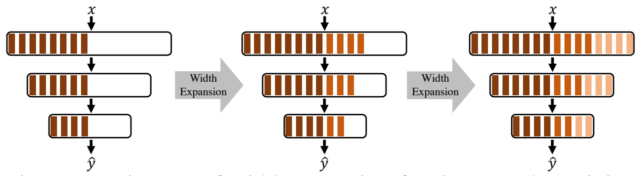


Figure 1. Diagram of width expansion for CNNs. The training is divided into 3 growth stages with $\alpha_0 = 0.5$, $\alpha_1 = 0.75$ and $\alpha_2 = 1$, respectively. Lighter color denotes the channels that later join the network.

finetuning. However, these one-shot pruning methods still spend more time on each forward-backward iteration to determine which neurons are important enough to be kept.

The main differences that separate our work from model pruning are two-fold. On the one hand, model pruning aims at accelerating inference speed and spends more time on training stage due to the need for finetuning after removing unimportant weights. However, our method directly reduces the training time and doesn't change the inference process. On the other hand, pruning algorithms find a sparse network by starting from a dense one, while our method eventually maintains a dense structure although starting training from a sparse one.

3. Method

3.1. Network Expansion

Let $f(\boldsymbol{x};\mathbf{w})$ be a deep neural network of any kind, where $\boldsymbol{x}$ denotes the input of f and $\mathbf{w}$ denotes the parameters of f. In order to succinctly parameterize the network architecture, we solely use a scalar variable $\alpha \in (0, 1]$ to control the scale of the network. Moreover, we define a model scaling operation $\otimes$ for parameter space to change the architecture of the model. Then we use $f(\boldsymbol{x};\alpha \otimes \mathbf{w})$ to denote the derived network whose structure is controlled by α. Obviously, the model architecture is shrunk when $\alpha < 1$ and is restored to the original size when $\alpha = 1$. When α gets increased, the model expanding operation is performed. Specifically, this will either introduce new channels to each convolutional layer from the perspective of width dimension, or add new layers to current network from the perspective of depth dimension. As a result, both the model architecture and its parameter space are enlarged. During the training process, if we gradually increase α from α_0 to 1, where $\alpha_0 \in (0, 1)$, an initially small sub-model $f(\boldsymbol{x};\alpha_0 \otimes \mathbf{w})$ can grow into a dense model $f(\boldsymbol{x};\mathbf{w})$ after multiple expansion operations. This means a great deal of FLOPs and memory access will be saved from both the forward-pass and backward-propagation computations. Therefore, the whole training process can be accelerated to save practical time cost and hardware resources. In the following section, we will describe how to accelerate the training process by using the network expansion strategy.

3.2. Width Expansion for CNNs

In a convolutional neural network (CNN) with L layers, the parameter set can be formulated as $\mathbf{w} = \{\mathbf{w}^l \in \mathbb{R}^{C_{l-1} \times C_l \times w \times h} | \, l = 1, 2, \cdots, L\}$. C_{l-1} and C_l denote the number of input and output feature maps of the l-th layer respectively, $C_0 = 3$ corresponds to the RGB channels of the input image $\boldsymbol{x}$, and (w, h) is the spatial size of the filter. The width scaling operation $\otimes$ changes the number of channels so that the shape of all weight matrices are scaled by $\alpha \otimes \mathbf{w}^l \in \mathbb{R}^{\lfloor \alpha C_{l-1} \rfloor \times \lfloor \alpha C_l \rfloor \times w \times h}$ for $\forall \, \mathbf{w}^l \in \mathbf{w}$.

Given the fact that CNN models are generally over-parameterized, many works [11, 37, 39, 50] have demonstrated a sparse sub-network can still reach the accuracy comparable to the original dense network and many channels in each layer can be taken away without harming the performance. Inspired by these works, we believe that a sparse network can serve as a good starting point for our training acceleration method. For a randomly initialized dense network $f(\boldsymbol{x};\mathbf{w})$, we treat $\alpha \in (0, 1]$ as the width factor, and α_0 the initial value. Before training, we keep only an α_0 ratio of channels for every convolutional layer and discard other channels along with the corresponding filter weights.

After the initial slimmed network $f(\boldsymbol{x};\alpha_0 \otimes \mathbf{w})$ is sampled from the dense $f(\boldsymbol{x};\mathbf{w})$, it will travel through n_g growth stages by conducting $n_g - 1$ times width expanding operation in order. A simple case is illustrated in Fig. 1 with $\alpha_0 = 0.5$ and $n_g = 3$. Suppose the network f is going to be trained for T epochs. In order to maintain the original training schedule, we equally split T epochs into n_g phase, which means the network $f(\boldsymbol{x};\alpha_i \otimes \mathbf{w})$ will be trained for T/n_g epochs every time after α gets increased.

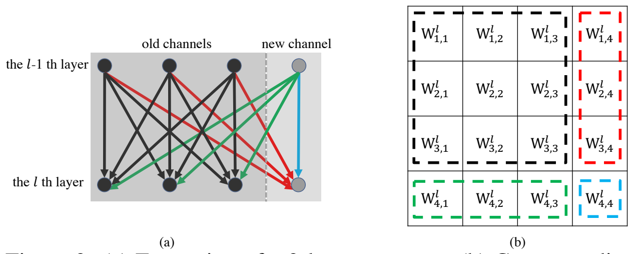
The main idea of our training acceleration method is to make a sparse model with less FLOPs gradually grow into a dense one during its training phase. Thus, we set different width factors α_i for each i-th stage to be linearly spaced between α_0 and α_{n_g-1}=1, and is calculated by

$$\alpha_i = \alpha_0 + i \cdot \frac{1-\alpha_0}{n_g - 1}, \quad 0 \le i \le n_g - 1 \tag{1}$$

where $i = \lfloor \frac{epoch}{T/n_g} \rfloor$ indicates the current growth stage. In addition, we use $\Delta\alpha = \alpha_{i+1} - \alpha_i$ to denote the expanding rate, which means a $\Delta\alpha$ proportion of the total channels in each layer will be added every time the network expands. It's worth noting that, without any modification to the optimizer and hyper-parameters such as learning rate and number of epochs, the only thing altered is the width (number of channels) of the network.

3.3. Weight Matrix Expanding Strategy

The newly grown channels in each layer lead to extra weights to be included in the convolutional filters. There is no doubt the later added weights go through less training



old channels
new channel
the l-1 th layer
the l th layer
(a)
$W^l_{1,1}$ $W^l_{1,2}$ $W^l_{1,3}$ $W^l_{1,4}$
$W^l_{2,1}$ $W^l_{2,2}$ $W^l_{2,3}$ $W^l_{2,4}$
$W^l_{3,1}$ $W^l_{3,2}$ $W^l_{3,3}$ $W^l_{3,4}$
$W^l_{4,1}$ $W^l_{4,2}$ $W^l_{4,3}$ $W^l_{4,4}$
(b)

Figure 2. (a) Expansion of a 2-layer conv net. (b) Corresponding weight matrix expanding diagram, each $W^l_{j,k}$ is a 2-D filter.

iterations, and therefore it would cause expressiveness mismatch between old and new feature maps within the same layer, which does harm to the performance of the final dense model. Thus, we cannot just randomly initialize the filter weights of new channels. In this work, we introduce an weight matrix expanding strategy to carefully expand the weight matrix $\mathbf{w}^l$ when α is enlarged. By imposing orthogonality across filters, the representative ability of the network is enhanced as the network grows [3, 41].

Firstly, we divide the filter weights of an expanding network into 4 types. If we treat a 2-D filter with the shape $w \times h$ as one entry of a matrix, then the weight tensor $\mathbf{w}^l$ can be view as a 2-D matrix of $\mathbb{R}^{C_{l-1} \times C_l}$, in which the (j, k) entry $\mathbf{w}^l_{j,k}$ is the filter that maps the j-th incoming feature map to the k-th outgoing feature map, where the index $j \in \{1, 2, \cdots, C_{l-1}\}$ and $k \in \{1, 2, \cdots, C_l\}$. As illustrated in Fig. 2 (a), the width scaling operation adds a new channel to both layers of a network with 3 channels. The black connections denote the old weights before expansion. Connections of other colors correspond to 3 types of new weights added. Red denotes old-to-new connections, green denotes new-to-old connections, blue denotes new-to-new connections. Fig. 2 (b) demonstrates the weight matrix $\mathbf{w}^l$ after width expansion, which is comprised of 4 sub-matrices each in a dashed box with corresponding color. For simplicity, we use $\boldsymbol{W}^o$, $\boldsymbol{W}^r$, $\boldsymbol{W}^g$, $\boldsymbol{W}^b$ to denote sub-matrices in the black, red, green, blue dashed box, respectively.

To ensure a good performance of the network, feature maps in the same layer are expected to encode different properties w.r.t. the input image to reduce information redundancy [31, 41]. Inspired by such designing principles, we propose to use filter orthogonality to reduce correlations between existing and newly-added feature maps. Taking Fig. 2 (b) as an example, when initializing $\boldsymbol{W}^g$, we expect $\boldsymbol{W}^g$ to be orthogonal to every row of $\boldsymbol{W}^o$ such that

$$\boldsymbol{W}^o_{row} \left(\boldsymbol{W}^g_{row}\right)^\top = [0, 0, 0]^\top \tag{2}$$

where $\boldsymbol{W}^o_{row} \in \mathbb{R}^{3 \times (3 \cdot w \cdot h)}$ and $\boldsymbol{W}^g_{row} \in \mathbb{R}^{1 \times (3 \cdot w \cdot h)}$ are the resulted matrices after flattening all entries of $\boldsymbol{W}^o$ and $\boldsymbol{W}^g$ into row vectors. Given the property of the nullspace of a random matrix $\boldsymbol{A} \in \mathcal{R}^{m \times n}$ $(m < n)$ that

$$\boldsymbol{A}\boldsymbol{x} = \mathbf{0}, \quad \forall \boldsymbol{x} \in \mathcal{N}(\boldsymbol{A}), \tag{3}$$

where $\mathcal{N}(\boldsymbol{A})$ is the nullspace of $\boldsymbol{A}$, we adopt singular value decomposition (SVD) to compute $\mathcal{N}(\boldsymbol{W}^o_{row})$ to find vectors orthogonal to the rows of $\boldsymbol{W}^o_{row}$:

$$\boldsymbol{W}^o_{row} = U \Sigma V^\top, \tag{4}$$

where $U \in \mathbb{R}^{3 \times 3}$, $\Sigma = diag(\sigma_1, \sigma_2, \sigma_3, 0, 0 \cdots) \in \mathbb{R}^{3 \times (3 \cdot w \cdot h)}$ and $V \in \mathbb{R}^{(3 \cdot w \cdot h) \times (3 \cdot w \cdot h)}$. The columns of V which correspond to zero singular values span the null space $\mathcal{N}(\boldsymbol{W}^o_{row})$, from which we can sample vectors orthogonal to the rows of $\boldsymbol{W}^o_{row}$ to initialize $\boldsymbol{W}^g$.

Similarly, we initialize $\boldsymbol{W}^r$ to be orthogonal to every column of $\boldsymbol{W}^o$ such that

$$\left(\boldsymbol{W}^o_{col}\right)^\top \boldsymbol{W}^r_{col} = [0, 0, 0]^\top \tag{5}$$

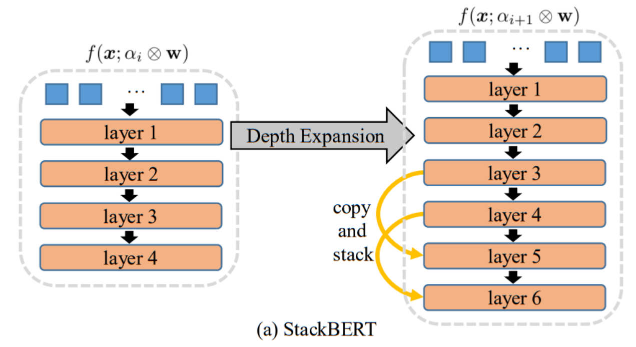
where $\boldsymbol{W}^o_{col} \in \mathbb{R}^{(3 \cdot w \cdot h) \times 3}$ and $\boldsymbol{W}^r_{col} \in \mathbb{R}^{(3 \cdot w \cdot h) \times 1}$ are the resulted matrices after flattening all entries of $\boldsymbol{W}^o$ and $\boldsymbol{W}^r$ into column vectors. This can be achieved by solving for $\mathcal{N}((\boldsymbol{W}^o_{col})^\top)$. Last but not least, we randomly initialize the filters in $\boldsymbol{W}^b$ based on [19], since these connections can be viewed as a sub-network within the whole network.

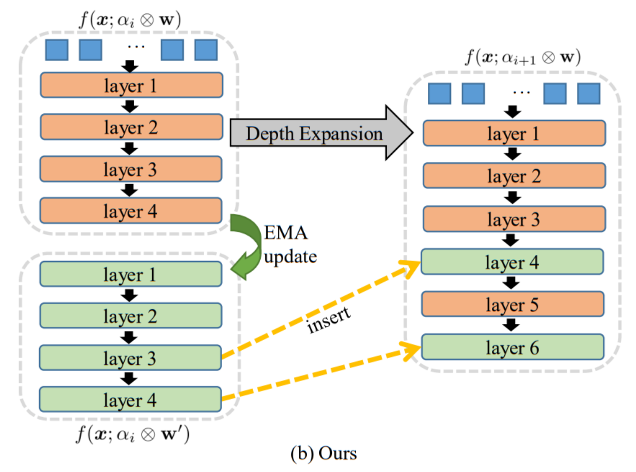
3.4. Depth Expansion for Vision Transformers

Besides CNNs, network expansion can also be used to accelerate the training of vision transformer. Considering that the transformer architecture [49] is stacked by multiple structurally identical blocks that conduct self-attention operation, we expand vision transformers by adjusting the number of attention blocks.

For a randomly initialized dense ViT model $f(\boldsymbol{x}; \mathbf{w})$ with L layers, where $\mathbf{w} = \{\mathbf{w}^l | \, l = 1, 2, \cdots, L\}$, we define $\alpha \in (0, 1]$ to be the depth factor that controls the number of layers. At the start of training, we keep only an α_0 ratio of the total layers to construct a shallowed ViT model $f(\boldsymbol{x}; \alpha_0 \otimes \mathbf{w})$ with $\alpha_0 \cdot L$ layers, and discard other layers. Similar to the width expansion for CNNs, this shallow ViT will be trained for n_g equi-length stages by conducting $n_g - 1$ times depth expanding operation to get deeper and deeper, while α grows from α_0 to $\alpha_{n_g-1} = 1$. During the i-th training stage, the depth factor $\alpha_i = \alpha_0 + i\frac{1-\alpha_0}{n_g-1}$ controls the number of layers $L_i = \alpha_i L$ of the current model.

As is shown in Fig. 3(a), previous works like StackBERT [13] directly copy the whole layers from the same trained model and stack them on the top of the network to conduct depth expansion. Although the model capacity is increased, such depth-expanding strategy will harm the model's expressiveness and cause significant accuracy drop in vision tasks. Since the new layers possess the same attention weights as those old ones being copied, redundancy is introduced to the expanded model. This is also what we try to avoid in the width expansion for CNNs. Therefore, we suggest that the newly added layers should perform mapping functions different from the old ones. To tackle this problem, we maintain an exponential moving





$f(\boldsymbol{x}; \alpha_i \otimes \mathbf{w})$
layer 1
layer 2
layer 3
layer 4
Depth Expansion
$f(\boldsymbol{x}; \alpha_{i+1} \otimes \mathbf{w})$
layer 1
layer 2
layer 3
layer 4
layer 5
layer 6
EMA update
insert
layer 1
layer 2
layer 3
layer 4
$f(\boldsymbol{x}; \alpha_i \otimes \mathbf{w}')$
(b) Ours

Figure 3. Diagram of depth expansion operation for ViT. For a 8-layer ViT, this figure demonstrates the situation where α is increased from $\alpha_i = 0.5$ to $\alpha_{i+1} = 0.75$. (a) StackBERT copies some layers from the same model and stacks them on top. (b) We propose to extract the layers from EMA model and insert them into the trained model to expand depth.

average (EMA) of the current trained model during each training stage as a parameter bank that provides new layers. During the i-th growth stage, the EMA model, denoted as $f(\boldsymbol{x}; \alpha_i \otimes \mathbf{w}')$, has the same architecture as the trained model, but its parameters are learned via the momentum update of $\alpha_i \otimes \mathbf{w}$ rather than forward- and backward- propagation. Specifically, supposing the final desired model $f(\boldsymbol{x}; \mathbf{w})$ has 8 layers, Fig. 3(b) shows the proposed depth expanding operation when the depth factor increases from $\alpha_i = 0.5$ to $\alpha_{i+1} = 0.75$. When entering the next training stage, $\Delta\alpha \cdot L$ attention blocks from the top of the EMA model will be extracted and inserted into the corresponding positions of the trained model to construct $f(\boldsymbol{x}; \alpha_{i+1} \otimes \mathbf{w})$, where $\Delta\alpha = \alpha_i - \alpha_{i-1}$ is the expanding rate. After the expanding operation, the EMA model will also be reinitialized. It's worth noting that the introduction of the EMA model helps to avoid the redundant parameters and the repeated attentions caused by directly copying. The extra time cost of momentum update can be almost omitted.

A unified accelerated training framework is described in Algorithm 1.

Algorithm 1 Procedure of the proposed training acceleration framework.

```
Require: a dense network $f(\boldsymbol{x}; \mathbf{w})$, total epochs $T$,
    initial scale factor $\alpha_0$, # of growth stages $n_g$,
    expanding dimension (width for CNN, depth for ViT).
Output: A trained network $f(\boldsymbol{x}; \mathbf{w})$.
 1: randomly initialize the dense network $f(\boldsymbol{x}; \mathbf{w})$;
 2: sample a sparse sub-network $f(\boldsymbol{x}; \alpha_0 \otimes \mathbf{w})$;
 3: for stage $i$ in $0, 1, \cdots, n_g - 1$ do
 4:     if $i > 0$ then
 5:         $\alpha_i \leftarrow \alpha_0 + i \cdot \frac{1-\alpha_0}{n_g - 1}$;
 6:         perform the network expanding operation
 7:         $f(\boldsymbol{x}; \alpha_i \otimes \mathbf{w}) \leftarrow f(\boldsymbol{x}; \alpha_{i-1} \otimes \mathbf{w})$;
 8:     end if
 9:     for epoch in $0, 1, \cdots, \frac{T}{n_g}$ do
10:         train the network $f(\boldsymbol{x}; \alpha_i \otimes \mathbf{w})$ for one epoch;
11:         validate the network;
12:     end for
13: end for
```

4. Experiments

4.1. Datasets And Implementations

In this section, we demonstrate the experiment results and ablation studies of the proposed training acceleration framework. We conduct experiments on three common classification benchmark: CIFAR10 [29], CIFAR100 and ImageNet-1k [9]. CIFAR10 contains a training set of 50K tiny-resolution RGB images that belong to 10 classes and a validation set of 10K images. CIFAR100 is a similar dataset with tiny RGB image but the number of classes is 100. ImageNet-1k is large-scale image dataset with 1,000 categories all from real-life scenario. This dataset is comprised of a training set of 1.28M samples and a validation set of 50K samples.

We choose ResNet [20] as our baseline model to validate the efficacy of our training acceleration method for CNN architecture, since it has both the block-stacking design and identity connections. This design paradigm is consistent with most of the modern CNN models. To guarantee fair comparison, we use Stochastic Gradient Descent (SGD) optimizer in all the experiments, and follow the standard data augmentation procedure, which includes random crop and random horizontal flip, proposed by original ResNet paper [20]. All experiments for ResNet are conducted with PyTorch [40] and the code is based on PyTorch official training scripts[1]. As for vision transformer, we choose DeiT-base model without the distilling token as the baseline. Since the original ViT [10] requires pretraining on the

[1]https://github.com/pytorch/examples/tree/main/imagenet

Model	Method	CIFAR10 Top1 Acc.(%)	CIFAR100 Top1 Acc.(%)	Wall-time consumption	Acceleration Ratio
ResNet32	baseline	94.41	74.91	1h : 29m	1.00×
	Expansion (ours)	94.36	74.87	1h : 03m	1.41×
ResNet56	baseline	95.01	75.80	2h : 31m	1.00×
	Expansion (ours)	94.88	75.76	1h : 44m	1.45×

Table 1. Top1 accuracy and acceleration on CIFAR10 and CIFAR100 datesets.

larger ImageNet-21k dataset, this undermines the fairness of the comparative experiments on practical time consumption. All the training setups and hyper-parameters are inherited from the official implementation[1].

Note that, all the wall-time consumption reported in the tables are the total training time which includes data-I/O, forward process, loss calculation, back-propagation of gradients, weight updates, and any other necessary training overheads such as maintaining the EMA model.

4.2. Result on CIFAR

On CIFAR datasets, we train all networks for 240 epochs with the batch size set to 128. The learning rate is decreased from 0.1 to 0 with a cosine decay schedule and the weight decay is set to 5e-4. The training time consumption is measured on a single NVIDIA-1080Ti GPU.

In order to strike a balance between the performance and acceleration ratio, we set the initial network width $\alpha_0 = 0.5$ and the number of growth stages $n_g = 5$. This leads to an expanding rate of $\Delta\alpha = \alpha_i - \alpha_{i-1} = 12.5\%$, which means 12.5% of the total channels will be added to each layer of the network after every 48 epochs when the width scaling operation is applied to expand the network. Note that, the performance gap can be mitigated by setting a bigger α_0, which leads to smaller acceleration ratio. This will be ablated later in Sec. 4.5. Under such network expansion setting, ResNet32 achieves 1.41× training acceleration with less than 0.05% accuracy decay on both dataset, ResNet56 achieves 1.45× acceleration with less than 0.13% accuracy decay on both dataset.

4.3. Result on ImageNet-1k

On ImageNet-1k, all ResNet models are trained for 120 epochs with the batch size set to 1024. The initial learning rate is empirically set to 0.4 to fit the batch size according to [28, 54] and is decreased to 0 with a cosine decay schedule. The weight decay is set to 1e-4. Considering the number of epochs for ImageNet is only half of that for the CIFAR dataset. We set the initial network width $\alpha_0 = 0.5$ and the number of growth stages $n_g = 3$. This means $\Delta\alpha$=25% of the total channels will be added to each layer of the network after every 40 epochs when the width scaling operation is applied to expand the network.

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

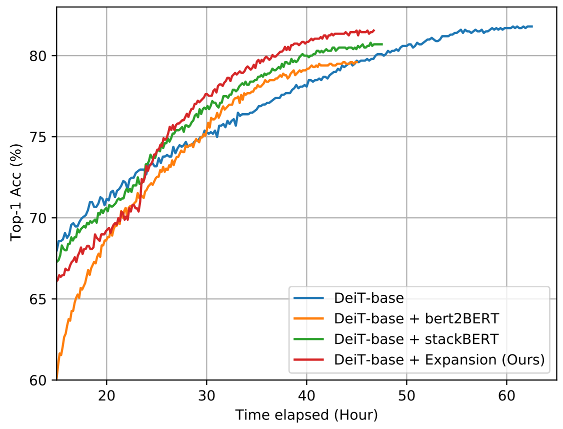


Figure 4. Top-1 accuracy for DeiT-base on ImageNet-1k validation set w.r.t. practical wall-time consumption.

The experiment results of ResNet50 and ResNet101 are shown in Tab. 2. For ResNet50 that is train on a cloud-service server with 4 TeslaV100 GPUs, we achieve 1.28× wall-time acceleration with a 0.3% performance drop. For ResNet101 that is trained with 8 TeslaV100 GPUs, we achieve 1.42× wall-time acceleration with a 0.2% performance drop. It's worth noting that our network expansion-based training acceleration method has better effect on heavy models with more parameters and layers. This is because the additional overhead such as memory access and kernel launching accounts for relatively less time in the whole training process of a bigger model.

For DeiT-base, we train it on a cloud-service server with 8 TeslaV100 GPUs for 300 epochs. We set batch size to be 1024, learning rate to be 1e-3 using a cosine scheduler with warmup. Optimizer is AdamW whose weight decay is 5e-2. In addition, we set initial depth factor $\alpha_0 = 0.5$, number of growth stage $n_g = 2$ for depth expansion. As a result, the proposed network expansion method achieves 81.5% top-1 accuracy on ImageNet (0.4% drop compared to baseline performance of DeiT-base), and accelerates the training process with 1.34× faster speed, and saves 15 hours. In practical uses, this means saving a 120-GPU-hour bill from a single run. The results of more expansion setups will be demonstrated and ablated in Sec. 4.5. Fig. 4 depicts the training curves of top1 validation accuracy w.r.t. wall-time consumption under different settings.

Model	Method	Top1 Acc.(%)	Wall-time consumption	Acceleration
ResNet50	baseline	76.2	19h : 21m	1.00×
	Net2Net [6]	75.6	15h : 10m	1.28×
	† Prunetrain [36]	74.8	-	1.32×
	Expansion (ours)	**75.9**	15h : 10m	1.28×
ResNet101	baseline	77.4	31h : 01m	1.00×
	Net2Net [6]	76.5	21h : 45m	1.42×
	Expansion (ours)	**77.2**	21h : 46m	1.42×
DeiT-base	baseline	81.8	62h : 30m	1.00×
	bert2BERT [5]	79.6	44h : 58m	1.39×
	StackBERT [13]	80.8	47h : 33m	1.31×
	Expansion (ours)	**81.5**	46h : 42m	1.34×

Table 2. Top1 accuracy and acceleration on ImageNet-1k. († data taken from the original paper)

Method	α_0	n_g	mIoU(%)	Time consumption	Acceleration
Baseline	1.0	1	76.9	21h : 59m	1.00×
Expansion	0.5	3	75.1	16h : 49m	1.30×
	0.5	2	76.4	16h : 50m	1.30×

Table 3. Results of DeepLabV3+ with ResNet101 backbone on Cityscapes dataset.

Method	α_0	n_g	Top1 Acc.	Time consumption	Acceleration
Baseline	1.0	1	75.80	2h : 31m	1.00×
Expansion	0.875	2	75.81	2h : 08m	1.18×
	0.75	3	75.88	1h : 59m	1.27×
	0.625	4	**75.91**	1h : 52m	1.35×
	0.5	5	75.76	1h : 44m	1.45×
	0.375	6	75.60	1h : 39m	1.52×
	0.25	7	75.53	1h : 36m	1.57×
	0.125	8	75.31	1h : 32m	1.64×

Table 4. Top1 accuracy and acceleration with different α_0 and n_g for ResNet56 on CIFAR100.

4.4. Downstream Task

We further transfer the proposed acceleration framework to downstream task semantic segmentation. We choose DeepLabV3+ as the baseline method since it uses ResNet-101 as feature extractor. We directly apply the network expansion method to its backbone network. We use 2,975 finely labeled images in the Cityscapes [8] training dataset and utilize the most basic data augmentation including random crop, color jitter and random horizontal flip. We use SGD optimizer with weight decay of 1e-4. The initial learning rate is 0.1 and is decayed with a poly schedule. We train DeepLabV3+ for 40,000 steps and set the output stride to be 16, batch size to be 16 to fit a single TeslaV100 GPU. We set the initial network width $\alpha_0 = 0.5$ and the number of growth stages $n_g = 3$. As shown in Tab. 3, the proposed method accelerates the training of DeepLabV3+ network for 1.3× faster with 0.5% mIoU drop.

4.5. Ablation Study

Trade-off between acceleration and performance. In the proposed training acceleration framework, the speedup ratio can be further boosted by setting a smaller initial width α_0. However, an overly shortened training process might eventually lead to acute performance drop. To verify the tradeoff relation between acceleration ratio and model performance, we conduct experiments by varying the initial network width α_0. The number of growth stages n_g varies accordingly since we fix the expanding rate $\Delta\alpha = 0.125$. Experiment results are shown in Tab. 4. In order to represent the relationship between speedup and performance more clearly and intuitively, we illustrate the same results in Fig. 5.

According to the curve in Fig. 5, the acceleration ratio and network accuracy are not simply in inverse proportion. When the speedup ratio is less than 1.34×, the model performance and acceleration ratio increase simultaneously. We conjecture that, with a large α_0 close to 1, the model capacity isn't reduced too much at the beginning of training, and therefore the final accuracy will not be constrained. Contrarily, our expanding training strategy serves as a regularization for the network, especially when CIFAR100 is a small dataset where a deep network is easy to overfit. When the speedup ratio is greater than 1.34×, decreasing α_0 will cause continuous performance degradation, although more training time can be saved. Based on the experiments, empirically setting $\alpha_0 = 0.5$ provides a good tradeoff for both performance and speedup.

We conduct similar experiments on DeiT-base and report the resulted accuracy and acceleration in Tab. 5. Generally, a smaller initial depth factor α_0 leads to more speedup ratio but more accuracy drop at the same time. From Tab. 5 we also find that too many times of network expansion operation is not suitable for vision transformer, which causes significant accuracy drop. Therefore, for a DeiT-base model with 12 layers, setting $\alpha_0 = 0.5$ and $n_g = 2$ provides the best trade-off.

Method	α_0	n_g	Top1 Acc.	Time consumption	Acceleration
Baseline	1.0	1	81.8	62h : 30m	1.00×
Expansion	$^1/_3$	3	80.2	42h : 27m	1.47×
	0.5	3	80.5	47h : 44m	1.31×
	0.5	2	81.5	46h : 42m	1.34×
	$^2/_3$	2	**81.8**	52h : 30m	1.19×

Table 5. Top1 accuracy and acceleration with different α_0 and n_g for DeiT-base on ImageNet-1k.

Method	Top1 Acc.
random	75.39
replicate	74.67
orthogonal	75.76

Table 6. Results of different weight matrix expanding methods for ResNet56 on CIFAR100.

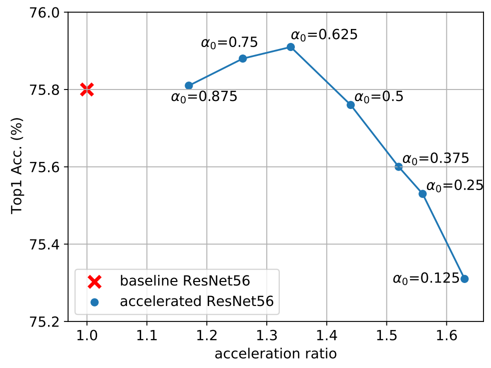
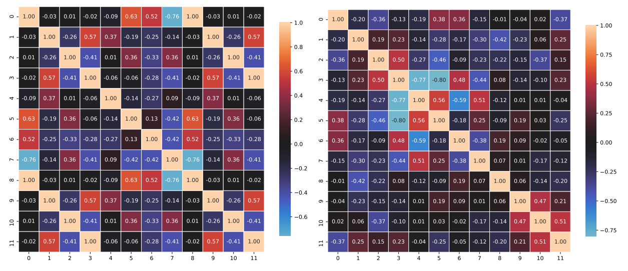


Figure 5. Trade-off between acceleration ratio and model performance by setting different α_0.



(a) (b)

Figure 6. Weights correlation between different convolution channels under (a) *replicate* expanding strategy, and (b) *orthogonal* expanding strategy.

Ablation study of weight matrix expanding strategy. We test different weight matrix expanding methods during the growth of network and show the results on CIFAR100 with ResNet56 in Tab. 6. *Random* means we sample random values for new weights from a normal distribution calculated by [19]. *Replicate* means we directly copy the weights from old filters to initialize the new ones. *Orthogonal* means we adopt the method described in Sec. 3.3. Experiment results show that *orthogonal* method outperforms the other two, and *replicate* method performs the worst. We conjecture that orthogonal filters generate new feature maps irrelevant to the old ones, which can enrich the expressive ability of the network. Conversely, replicated filters tend to produce similar feature maps that may limit the development of model's representative ability.

We further visualize the correlation between feature maps of the same layer after width expansion to support our conjecture and show the results in Fig. 6a and Fig. 6b. Specifically, for a ResNet20 that has just been expanded, we choose the 3rd convolutional layer, whose C_3 is increased from 8 to 12. We then randomly sample an image from the validation set of CIFAR100 and obtain the corresponding output feature maps. The Pearson correlation coefficient [4] is calculated between each pair of the 12 feature maps. Since we copy the upper-left corner of $\boldsymbol{W}^o$ (please refer to Sec. 3.3) to initialize the newly grown weights, as shown in Fig. 6a, the 1st, 2nd, 3rd and 4th channels are directly correlated with the 9th, 10th, 11th and 12th channels, respectively. Oppositely, when applying orthogonal weight matrix expanding method, as shown in Fig. 6b, the feature maps have low cross-channel correlations as expected.

5. Conclusion

This paper proposes a practical training acceleration method via network expansion for deep neural networks, which can be easily integrated into popular deep learning frameworks on general GPU devices. Specifically, our method takes the advantage of the sparsity of deep neural networks in both width and depth dimension. We further introduce a weight matrix expanding strategy for CNNs via orthogonal filters to help increase the network's representative ability and lower the information redundancy across channels. We also design a depth expanding strategy for ViTs by using the corresponding EMA model to avoid redundancy caused by direct copy. Extensive experiments demonstrate the effectiveness of our method. We achieve ideal acceleration results with insignificant performance drop on several tasks for both CNN and ViT architectures. Our training acceleration frameworks can reduce the practical time and power consumption caused by training deep networks, which is also beneficial and cost-efficient to those cloud service users who pay bills by hours. Thus, this work is of great significance in a practical sense.

6. Acknowledgments

This work is supported by National Key R&D Program of China under Grant No.2022ZD0160304. We gratefully acknowledge the support of MindSpore [25], CANN (Compute Architecture for Neural Networks) and Ascend AI Processor used for this research.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016. 2

[2] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017. 2

[3] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep networks? *Advances in Neural Information Processing Systems*, 31, 2018. 4

[4] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009. 8

[5] 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 *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2134–2148, 2022. 2, 7

[6] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *International Conference on Learning Representations (ICLR)*, 2016. 2, 7

[7] Tianyi Chen, Bo Ji, Tianyu Ding, Biyi Fang, Guanyi Wang, Zhihui Zhu, Luming Liang, Yixin Shi, Sheng Yi, and Xiao Tu. Only train once: A one-shot neural network training and pruning framework. *Advances in Neural Information Processing Systems*, 34, 2021. 2

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. 7

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 5

[10] 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. *arXiv preprint arXiv:2010.11929*, 2020. 5

[11] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018. 3

[12] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 1

[13] Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of bert by progressively stacking. In *International conference on machine learning*, pages 2337–2346. PMLR, 2019. 2, 4, 7

[14] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 2

[15] Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. Vision gnn: An image is worth graph of nodes. *arXiv preprint arXiv:2206.00272*, 2022. 1

[16] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *Advances in Neural Information Processing Systems*, 34:15908–15919, 2021. 1

[17] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. *International Conference on Learning Representations (ICLR)*, 2017. 2

[18] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015. 2

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 4, 8

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5

[21] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017. 2

[22] Peng Hu, Xi Peng, Hongyuan Zhu, Mohamed M Sabry Aly, and Jie Lin. Opq: Compressing deep neural networks with one-shot pruning-quantization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7780–7788, 2021. 2

[23] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 1

[24] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019. 1

[25] Huawei. Mindspore. `https://www.mindspore.cn/`, 2020. 8

[26] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018. 2

[27] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pages 491–507. Springer, 2020. 1

[28] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014. 6

[29] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012. 1

[31] Haoyu Liang, Zhihao Ouyang, Yuyuan Zeng, Hang Su, Zihao He, Shu-Tao Xia, Jun Zhu, and Bo Zhang. Training interpretable convolutional neural networks by differentiating class-specific filters. In *European Conference on Computer Vision*, pages 622–638. Springer, 2020. 4

[32] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 1

[33] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021. 2

[34] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022. 1

[35] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 1

[36] Sangkug Lym, Esha Choukse, Siavash Zangeneh, Wei Wen, Sujay Sanghavi, and Mattan Erez. Prunetrain: fast neural network training by dynamic sparse model reconfiguration. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2019. 1, 2, 7

[37] Xiaolong Ma, Geng Yuan, Xuan Shen, Tianlong Chen, Xuxi Chen, Xiaohan Chen, Ning Liu, Minghai Qin, Sijia Liu, Zhangyang Wang, et al. Sanity checks for lottery tickets: Does your winning ticket really win the jackpot? *Advances in Neural Information Processing Systems*, 34, 2021. 3

[38] Lu Miao, Xiaolong Luo, Tianlong Chen, Wuyang Chen, Dong Liu, and Zhangyang Wang. Learning pruning-friendly networks via frank-wolfe: One-shot, any-sparsity, and no retraining. In *International Conference on Learning Representations*, 2021. 2

[39] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *Advances in neural information processing systems*, 32, 2019. 3

[40] 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. *Advances in neural information processing systems*, 32, 2019. 2, 5

[41] Aaditya Prakash, James Storer, Dinei Florencio, and Cha Zhang. Repr: Improved training of convolutional filters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10666–10675, 2019. 4

[42] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, 2019. 1

[43] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 1

[45] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1

[46] Yehui Tang, Kai Han, Yunhe Wang, Chang Xu, Jianyuan Guo, Chao Xu, and Dacheng Tao. Patch slimming for efficient vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12165–12174, 2022. 2

[47] Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chunjing Xu, Chao Xu, and Chang Xu. Scop: Scientific control for reliable neural network pruning. *Advances in Neural Information Processing Systems*, 33:10936–10947, 2020. 2

[48] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021. 1

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 4

[50] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2019. 3

[51] 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. *Advances in Neural Information Processing Systems*, 32, 2019. 1, 2

[52] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34:12077–12090, 2021. 1

[53] Xucheng Ye, Pengcheng Dai, Junyu Luo, Xin Guo, Yingjie Qi, Jianlei Yang, and Yiran Chen. Accelerating cnn training by pruning activation gradients. In *European Conference on Computer Vision*, pages 322–338. Springer, 2020. 1, 2

[54] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10, 2018. 2, 6

[55] Jiong Zhang, Hsiang-Fu Yu, and Inderjit S Dhillon. Autoassist: A framework to accelerate training of deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019. 1, 2

[56] Xiao Zhou, Weizhong Zhang, Zonghao Chen, Shizhe Diao, and Tong Zhang. Efficient neural network training via forward and backward propagation sparsification. *Advances in Neural Information Processing Systems*, 34, 2021. 2

[57] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020. 1