Network Expansion For Practical Training Acceleration

Ning Ding\textsuperscript{1,2}, Yehui Tang\textsuperscript{1,2}, Kai Han\textsuperscript{2*}, Chao Xu\textsuperscript{1}, Yunhe Wang\textsuperscript{2*}

\textsuperscript{1}National Key Lab of General AI, School of Intelligence Science and Technology, Peking University
\textsuperscript{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-Ik). 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 demonstrated 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 general GPU platforms. [33] 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
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 $1.42 \times$ wall-time acceleration for training ResNet-101, $1.34 \times$ 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 pretrained 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 backpropagation 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
3. Method

3.1. Network Expansion

Let \( f(x; w) \) be a deep neural network of any kind, where \( x \) denotes the input of \( f \) and \( w \) 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 \( f(x; \alpha \otimes w) \) to denote the derived network whose structure is controlled by \( \alpha \). Obviously, the model architecture is shrunk when \( \alpha < 1 \) and is restored to the original size when \( \alpha = 1 \). When \( \alpha \) 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 \( \alpha \) from \( \alpha_0 \) to 1, where \( \alpha_0 \in (0, 1) \), an initially small sub-model \( f(x; \alpha_0 \otimes w) \) can grow into a dense model \( f(x; w) \) 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 \( w = \{ w^l \mid w^l \in \mathbb{R}^{C_{l-1} \times C_l \times w \times h} \} \). \( C_{l-1} \) 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 \( x \), 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 \( \alpha \otimes w^l \in \mathbb{R}^{[\alpha C_{l-1}] \times \alpha C_l \times w \times h} \) for all \( w^l \).

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.

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 \( f(x; w) \), we treat \( \alpha \in (0, 1) \) as the width factor, and \( \alpha_0 \) the initial value. Before training, we keep only an \( \alpha_0 \) ratio of channels for every convolutional layer and discard other channels along with the corresponding filter weights.

After the initial slimmer network \( f(x; \alpha_0 \otimes w) \) is sampled from the dense \( f(x; w) \), 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 \( f(x; \alpha_i \otimes w) \) will be trained for \( T/n_g \) epochs every time after \( \alpha \) 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 \( \alpha_i \) for each \( i \)-th stage to be linearly spaced between \( \alpha_0 \) and \( \alpha_{n_g - 1} = 1 \), and is calculated by

\[
\alpha_i = \alpha_0 + i \cdot \frac{1 - \alpha_0}{n_g - 1}, \quad 0 \leq i \leq n_g - 1
\]

where \( i = \lfloor \frac{\text{epoch}}{T/n_g} \rfloor \) 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
A random matrix into row vectors. Given the property of the nullspace of resulted matrices after flattening all entries of $W$, we propose to use filter orthogonality to reduce correlations and redundancy \cite{31, 41}. Inspired by such designing principles, maps in the same layer are expected to encode different ity, we use each in a dashed box with corresponding color. For simplicity after width expansion, which is comprised of 4 sub-matrices denoted new-to-old connections, blue denotes new-to-new connections. Connections of other colors correspond to 3 types of new channel to both layers of a network with 3 channels. The treated in Fig. 2 (a), the width scaling operation adds a new feature map to the $l$-th layer of a model. Thus, we cannot just randomly initialize the filter layer, which does harm to the performance of the final dense model. Therefore, we suggest that the newly added layers should per- form mapping functions different from the old ones. To ensure a good performance of the network, feature weights of an expanding net- work to be orthogonal to every row of $W$, from which we can sample vectors orthogonal to the rows of $W_{row}^\alpha$ to initialize $W_{col}^g$.

Similarly, we initialize $W^{\tau}$ to be orthogonal to every column of $W^\alpha$ such that

$$ (W_{col}^\alpha)^T W_{col} = [0, 0, 0]^T \quad (5) $$

where $W_{col}^\alpha \in \mathbb{R}^{(3\times w-h)\times 3}$ and $W_{col}^g \in \mathbb{R}^{(3\times w-h)\times 1}$ are the resulted matrices after flattening all entries of $W^\alpha$ and $W^{\tau}$ into column vectors. This can be achieved by solving for $N((W_{col}^\alpha)^T)$. Last but not least, we randomly initialize the filters in $W_b$ based on \cite{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 \cite{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 $f(x; w)$ with $L$ layers, where $W = \{w_l | l = 1, 2, \cdots, L\}$, 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 $\alpha_0$ ratio of the total layers to construct a shallowed ViT model $f(x; \alpha_0 \otimes w)$ 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 $\alpha$ grows from $\alpha_0$ to $\alpha_{n_g - 1} = 1$. During the $i$-th training stage, the depth factor $\alpha_i = \alpha_0 + \frac{i - \frac{n_g + 1}{2}}{n_g - 1}$ controls the number of layers $L_i = \alpha_i L$ of the current model.

As shown in Fig. 3(a), previous works like Stack-BERT \cite{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
8-layer ViT, this figure demonstrates the situation where $\alpha$ increased from where $\Delta \alpha$ positions of the trained model to construct model will be extracted and inserted into the corresponding model helps to avoid the redundant parameters and the reinitialized. It’s worth noting that the introduction of the EMA expanding operation, the EMA model will also be reinitialized stage,

\[ \alpha_i \leftarrow \alpha_0 + i \cdot \frac{1 - \alpha_0}{n_g - 1} \]

\[ f(x; \alpha_i \otimes w) \leftarrow f(x; \alpha_{i-1} \otimes w) \]

since the original ViT requires pretraining on the DeiT-base model without the distilling token as the base.

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. As for vision transformer, we choose DeiT-base model without the distilling token as the baseline. Since the original ViT requires pretraining on the

Figure 3. Diagram of depth expansion operation for ViT. For a 8-layer ViT, this figure demonstrates the situation where $\alpha$ 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 $f(x; \alpha_i \otimes w')$, has the same architecture as the trained model, but its parameters are learned via the momentum update of $\alpha_i \otimes w$ rather than forward- and backward-propagation. Specifically, supposing the final desired model $f(x; w)$ 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 $f(x; \alpha_{i+1} \otimes w)$, where $\Delta \alpha = \alpha_{i+1} - \alpha_i$ 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 $f(x; w)$, 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 $f(x; w)$.

1. randomly initialize the dense network $f(x; w)$;
2. sample a sparse sub-network $f(x; \alpha_0 \otimes w)$;
3. for stage $i$ in $0, 1, \ldots, n_g - 1$ 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. $f(x; \alpha_i \otimes w) \leftarrow f(x; \alpha_{i-1} \otimes w)$;
   8. end if
   9. for epoch in $0, 1, \ldots, \frac{T - f}{f}$ do
   10. train the network $f(x; \alpha_i \otimes w)$ for one epoch;
   11. validate the network;
   12. end for
4. 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. As for vision transformer, we choose DeiT-base model without the distilling token as the baseline. Since the original ViT requires pretraining on the

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

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 $a_0 = 0.5$ 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 $a_0$, 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 $a_0 = 0.5$ 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.

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 $a_0 = 0.5$, 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.

### Table 1. Top1 accuracy and acceleration on CIFAR10 and CIFAR100 datasets.

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

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

1https://github.com/facebookresearch/deit
### 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 $\alpha_0 = 16$, batch size to be 16 to fit a single TeslaV100 GPU.

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. 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×, the model performance will not be constrained.

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. 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×, the model performance will not be constrained.

### 4.5. Ablation Study

We conduct similar experiments on DeiT-base and report the results in Tab. 5. Generally, a smaller initial depth factor $\alpha_0$ 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.
<table>
<thead>
<tr>
<th>Method</th>
<th>( \alpha_0 )</th>
<th>( n_g )</th>
<th>Top1 Acc.</th>
<th>Time consumption</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>1.0</td>
<td>1</td>
<td>81.8</td>
<td>62h : 30m</td>
<td>1.00×</td>
</tr>
<tr>
<td>Expansion</td>
<td>( \frac{7}{8} )</td>
<td>3</td>
<td>80.2</td>
<td>42h : 27m</td>
<td>1.47×</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>3</td>
<td>80.5</td>
<td>47h : 44m</td>
<td>1.31×</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>2</td>
<td>81.5</td>
<td>46h : 42m</td>
<td>1.34×</td>
</tr>
<tr>
<td></td>
<td>( \frac{7}{8} )</td>
<td>2</td>
<td>81.8</td>
<td>52h : 30m</td>
<td>1.19×</td>
</tr>
</tbody>
</table>

Table 5. Top1 accuracy and acceleration with different \( \alpha_0 \) and \( n_g \) for DeiT-base on ImageNet-1k.

<table>
<thead>
<tr>
<th>Method</th>
<th>Top1 Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>random</td>
<td>75.39</td>
</tr>
<tr>
<td>replicate</td>
<td>74.67</td>
</tr>
<tr>
<td>orthogonal</td>
<td>75.76</td>
</tr>
</tbody>
</table>

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

**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 \( W^o \) (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


[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


