

UPDP: A Unified Progressive Depth Pruner for CNN and Vision Transformer

Ji Liu*, Dehua Tang*, Yuanxian Huang, Li Zhang, Xiaocheng Zeng, Dong Li, Mingjie Lu, Jinzhang Peng, Yu Wang, Fan Jiang, Lu Tian, Ashish Sirasao

Advanced Micro Devices, Inc., Beijing, China

{liuji, dehua.tang, yuanxian, xze, d.li, mingjiel, jinz.peng, yu.w, f.jiang, lu.tian, ashish.sirasao}@amd.com

Abstract

Traditional channel-wise pruning methods by reducing network channels struggle to effectively prune efficient CNN models with depth-wise convolutional layers and certain efficient modules, such as popular inverted residual blocks. Prior depth pruning methods by reducing network depths are not suitable for pruning some efficient models due to the existence of some normalization layers. Moreover, finetuning subnet by directly removing activation layers would corrupt the original model weights, hindering the pruned model from achieving high performance. To address these issues, we propose a novel depth pruning method for efficient models. Our approach proposes a novel block pruning strategy and progressive training method for the subnet. Additionally, we extend our pruning method to vision transformer models. Experimental results demonstrate that our method consistently outperforms existing depth pruning methods across various pruning configurations. We obtained three pruned ConvNeXtV1 models with our method applying on ConvNeXtV1, which surpass most SOTA efficient models with comparable inference performance. Our method also achieves state-of-the-art pruning performance on the vision transformer model.

Introduction

Deep neural networks (DNNs) have made significant strides across various tasks, culminating in remarkable successes within industrial applications. Among these applications, the pursuit of model optimization stands out as a prevalent need, offering the potential to elevate model inference speed while minimizing accuracy trade-offs. This pursuit encompasses a range of techniques, notably model pruning, quantization, and efficient model design. The efficient model design includes neural architecture search (NAS) (Cai et al. 2020; Yu and Huang 2019; Yu et al. 2020; Wang et al. 2021a) and handcraft design methodologies. Model pruning has emerged as a prevalent strategy for optimizing models in industrial applications. Serving as a primary acceleration approach, model pruning focuses on the deliberate removal of redundant weights while maintaining accuracy. This process typically involves three sequential steps: initial baseline model training, subsequent pruning of less vital

*These authors contributed equally.

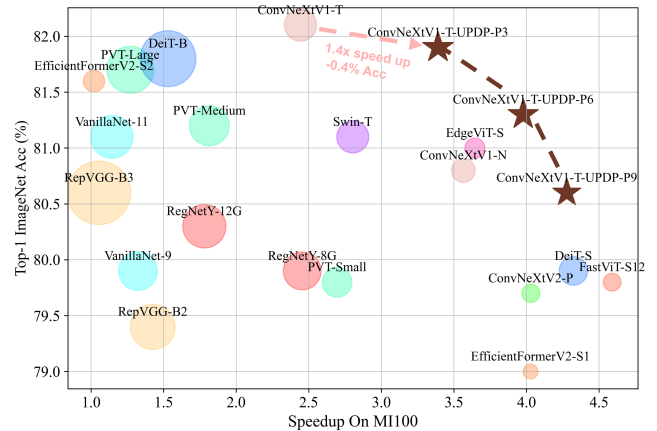


Figure 1: Performance vs. speedup on the ImageNet-1K. Our three pruned ConvNeXtV1 models surpass most SOTA efficient models on performance including RegNetY, RepVGG, VanillaNet, ConvNeXtV2, Swin-T, PVT, DeiT, EdgeViT, EfficientFormerV2, and FastViT.

weights or layer channels, and a concluding finetuning phase for the pruned model. Notably, model pruning can be classified into two categories: non-structured pruning and structured pruning. Structured pruning is the preferred approach for model deployment in industrial applications, primarily due to hardware limitations. In contrast to non-structured methods, where less important weights in convolutional kernel layers are zeroed out in a sparse manner within each kernel channel, structured pruning encompasses techniques like channel-wise pruning and block pruning. Channel-wise pruning focuses on eliminating entire channel filters within the kernel, while block pruning operates at a larger scale, typically targeting complete blocks. Given that block pruning often leads to a reduction in model depth, it is also referred to as a depth pruner.

The evolution of CNN model design has led to the development of more efficient models. For instance, MobileNetV2 (Sandler et al. 2018) employs numerous depth-wise convolutional layers and stacks inverted residual blocks, achieving high performance while minimizing parameters and flops. ConvNeXtV1 (Liu et al. 2022) leverages

the large kernel trick and incorporates stacked inverted residual blocks to achieve remarkable efficiency. The conventional channel-wise pruning method faces challenges with depth-wise convolutional layers due to sparse computation and fewer parameters. Moreover, now model platforms favor a higher degree of parallel computing like GPUs, and channel-wise pruning methods would make efficient models thinner and sparser, which leads to low hardware utilization and thus inferior achievable hardware efficiency. To address these issues, DepthShrinker (Fu et al. 2022) and Layer-Folding (Dror et al. 2021) are proposed to optimize MobileNetV2 by reducing model depth through reparameterization techniques (Ding et al. 2021a,b). However, these methods exhibit certain limitations. (1) The mechanism of finetuning subnet with removing activation layers directly could potentially compromise the integrity of baseline model weights, hindering the attainment of high performance. (2) These methods come with usage constraints, they are unable to prune models with some normalization layers like LayerNorm (Ba, Kiros, and Hinton 2016) or GroupNorm (Wu and He 2018) layer, because reparameterization technique cannot merge normalization layer which is not BatchNorm layer into adjacent convolutional layer or full-connection layer. (3) These methods cannot be applied to vision transformer models for optimization due to the existence of LayerNorm layer.

To alleviate these problems, we propose a progressive training strategy and novel block pruning method for our depth pruning approach that can prune both CNN and vision transformer models. The progressive training strategy can smoothly transfer the baseline model structure to the subnet structure with high utilization of baseline model weights, which leads to higher accuracy. Our proposed block pruning method can handle the existing normalization layer issue, which can handle all activation and normalization layers in theory. Thus, our method can prune vision transformer models, which is not suitable for existing depth pruning methods. Our experimental evaluation spans across ResNet34, MobileNetV2, and ConvNeXtV1, showcasing the superior pruning capabilities. As shown in Figure 1, pruned ConvNeXtV1 models with our method surpass most SOTA efficient models with comparable inference performance. Notably, we extend our exploration to vision transformer models, achieving leading pruning results compared to other vision transformer pruning methods.

Our main contributions can be summarized as follows. (1) We propose a unified and efficient depth pruning method for optimizing both CNN and vision transformer models. (2) We propose a progressive training strategy for subnet optimization, coupled with a novel block pruning strategy using reparameterization technique. (3) Conducting comprehensive experiments on both CNN and vision transformer models to showcase the superior pruning performance of our depth pruning method.

Related Work

Network Pruning. Pruning algorithms can be roughly divided into two types. One is the non-structured pruning algorithm represented by (Han, Mao, and Dally 2015; Elsen

et al. 2020; Pool and Yu 2021). It removes redundant elements in the weight according to certain criteria. However, non-structured pruning requires special software or hardware accelerators for the pruned models, so its versatility is not strong. In contrast to unstructured pruning, structured pruning prunes the entire parameter structure, such as discarding entire rows or columns of weights, or entire filters in convolutional layers.

When VGG (Simonyan and Zisserman 2014) and ResNet (He et al. 2016) were on the rise, Pruning Filters (Li et al. 2016) adopts the L1-norm to select unimportant channels and prune them. Network FPGM (He et al. 2019) utilizes the geometric median of the convolutional filter to find redundant filters. Subsequently, various efficient DNN networks, such as MobileNet and its variants (Howard et al. 2017, 2019; Tan et al. 2019; Radosavovic et al. 2020), incorporated depthwise convolutions (Chollet 2017) to accelerate speed and improve accuracy, enabling real-time deployment on diverse hardware platforms. MatePruning (Liu et al. 2019) proposes the concept of PruningNet, which automatically generates weights for the pruned model, thus avoiding retraining. However, while depthwise convolution offers advantages in terms of reduced computation and parameters, it also presents a drawback—an increased memory footprint, posing a challenge for computationally intensive hardware like GPUs and DSPs (Tan and Le 2021). Unfortunately, channel-wise pruning methods do not offer an intuitive and efficient solution to address this memory footprint challenge.

The most relevant to our work is layer-wise pruning, which can completely remove a block or layer to reduce the depth of the network and effectively alleviate the problem of memory usage. Shallowing deep networks (Chen and Zhao 2018) and LayerPrune (Elkerdawy et al. 2020) propose their own strategies for evaluating the importance of convolutional layers. ESNB (Zhou, Yen, and Yi 2021) and ResConv (Xu et al. 2020) identify which layers to be pruned by evolutionary search algorithms and differentiable parameters, respectively. Layer-Folding (Dror et al. 2021) and DepthShrinker (Fu et al. 2022) remove non-linear activation functions within the block and merge multiple layers into a single layer using structural reparameterization techniques. Layer-Folding and DepthShrinker have only been verified on the few limited models, and the hard removal of ReLU may have an impact on the accuracy of the subnet.

The Transformer family of models excels in performance across various vision tasks (Carion et al. 2020; Strudel et al. 2021; Brown et al. 2020); however, its high inference cost and significant memory footprint hinder widespread adoption (Pope et al. 2023). To tackle the memory footprint challenge, layer-wise pruning presents an effective solution. Dynamic skipping blocks to remove some layers has become the mainstream transformer compression method (Zhang and He 2020; Dong, Cordonnier, and Loukas 2021; Michel, Levy, and Neubig 2019). DynamicViT (Rao et al. 2021) dynamically screens the number of tokens that need to be passed to the next layer. By encouraging dimension-wise sparsity, VTP (Zhu, Tang, and Han 2021) selects the dimension with strong redundancy for pruning.

Structural Reparameterization. In the absence of a nonlinear activation function within a block, the structural reparameterization technique facilitates the consolidation of multiple convolutional layers into a single convolutional layer (Bhardwaj et al. 2022). This consolidation effectively diminishes the neural network’s memory requirements during inference, resulting in accelerated model processing. RepVGG (Ding et al. 2021b) distinguishes between training and testing structures, empowering the plain network to surpass the performance of ResNet. Furthermore, DBB (Ding et al. 2021a) merges a multi-branch architecture into a single convolution, significantly outpacing the speed of a conventional multi-branch unit.

Neural Architecture Search (NAS). Weight-sharing NAS has become the mainstream of pruning methods due to its flexibility and convenience of training a supernet and deploying multiple subnets. Once-for-All (Cai et al. 2020) uses a progressive training supernet. BigNAS (Yu et al. 2020) uses a series of simple and practical training methods to improve the efficiency of training supernet. Once the supernet is trained, typical search algorithms, such as genetic search, can be applied to find a set of Pareto-optimal networks for various deployment scenarios.

In this work, we propose a unified depth pruning approach for both efficient CNN and vision transformer models with a progressive training strategy, a novel block pruning method, and the reparameterization technique. Differing from DepthShrinker and Layer-Folding finetuning the subnet with direct activation layer removal, our method progressively removes the activation layer in the pruned block during subnet training. Moreover, our method handles the normalization layer problem that DepthShrinker and Layer-Folding cannot prune models with LayerNorm or Group-Norm layers in the block. Further, they can not prune vision transformer models. Although our work has a similar training process as VanillaNet (Chen et al. 2023), whereas VanillaNet is proposed to design a completely new network structure, our method is a general depth pruning framework for both CNN and vision transformer models.

Method

Unified Progressive Depth Pruner

Our depth pruning approach aims to reduce model depth by proposed novel block pruning strategy with reparameterization technique rather than directly omitting the block. As shown in Figure 2, our block pruning strategy converts a complex and slow block into a simple and fast block in block merging. For a block, we replace the activation layer with identity layer and replace the LayerNorm (LN) or Group-Norm (GN) layer with a BatchNorm (BN) layer and insert an activation layer with a BatchNorm layer at the end of block to create conditions for reparameterization. Then, the reparameterization technique can merge the BatchNorm layers, adjacent Convolutional or Full-connection layers and skip connections as shown in Figure 2.

Overview. Our approach primarily consists of four main steps, which are supernet training, subnet searching, subnet training, and subnet merging. First, We construct a supernet

based on the baseline model, where we make block modification as shown in Figure 2. After supernet training, a search algorithm is used to search an optimal subnet. Then, we adopt a proposed progressive training strategy to optimize the optimal subnet with less accuracy loss. In the end, the subnet would be merged into a shallower model with the reparameterization technique.

Supernet Training. Efficient CNN and vision transformer models usually consist of several basic blocks, like some efficient models structure shown in Figure 2. First, we construct a supernet based on the baseline model and then train a robust supernet model based on the sandwich rule (Yu and Huang 2019) method to ensure that each subnet has a meaningful accuracy. We combine the baseline block and corresponding pruned block into a supernet block, which has both baseline block and pruned block flows. For a supernet block, choosing the baseline block flow means no pruning and choosing pruned block flow means pruning the block. Then, subnet selection is a series of choices where the choice number is equal to a block number of the baseline model. The subnet would be faster with more pruned blocks selected.

Inspired by BigNAS (Yu et al. 2020), we adopt the sandwich rule to sample the subnetwork before each step. In each step, we sample four sequential subnets, where the first one keeps all blocks unpruned, the following two subnets randomly select blocks to be pruned, and the last one keeps all blocks pruned. Then, we optimize supernet with accumulated grads of four subnets. The sandwich rule can effectively guarantee the upper and lower limits of the trained supernet. Also many methods (Wang et al. 2021b,a) demonstrate that the sandwich rule can be used to train supernet efficiently, even if the number of epochs is small and the accuracy distribution of the subnet is the same as that of training more epochs. In this way, we can reduce the training cost of supernet.

Subnet Searching. The primary objective of our depth pruner is to identify an optimal subnet based on a specified pruning criteria, such as the number of blocks to be pruned. As shown in Equation 1, we formulate this problem as an optimal problem. For all samples X and their labels Y , the goal of subnet searching is to find a subnet S_p with the highest accuracy. $p \in R^{N_{block}}$ is a binary vector, representing the pruning setting of the subnet. If i -th block is pruned, p_i is set to 1. The number of pruned blocks of each subnet is equal to k . Genetic algorithm (Cai et al. 2020) is applied to solve this problem.

$$\arg \max_p \text{Accuracy}(S_p(X), Y) \quad \text{s.t.} \quad \|p\|_0 = k \quad (1)$$

After the search process, we obtain a subnet that has a specified number of pruned blocks, and other blocks keep the same as the baseline model.

Subnet Training. We need to train the optimal subnet obtained from the previous step to restore its accuracy. Rather than directly training the subnet, our approach employs a progressive training strategy to finetune the subnet smoothly transferring from baseline model weights. The subnet training consists of two stages. During the first training stage,

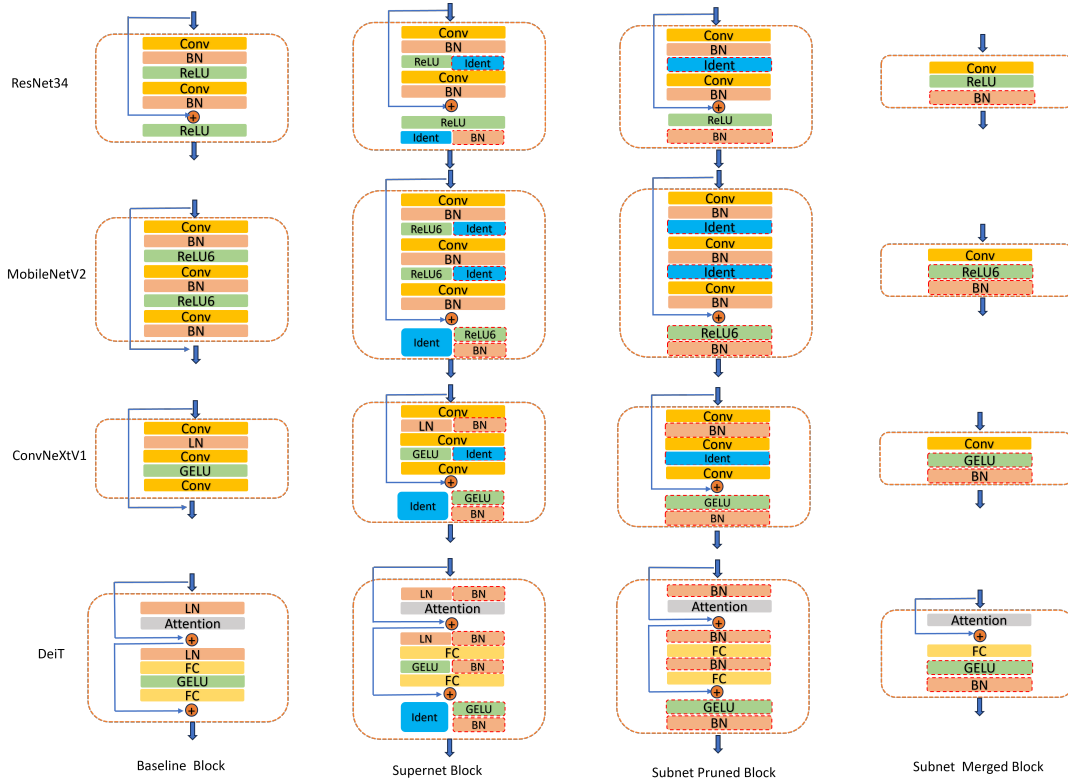


Figure 2: Framework overview of our proposed depth pruner. Each pruned baseline block will gradually evolve into a smaller merged block to speedup and save memory. Four baselines are experimented, including three CNN-based networks (ResNet34, MobileNetV2 and ConvNeXtV1) and one vision transformer network (DeiT-Tiny).

we adopt a progressive training strategy, gradually transferring from a baseline model structure to the pruned subnet structure by a controlling λ factor. In the second stage, we continue finetuning the pruned subnet to the end for high accuracy.

As for the first stage, the gradual transition of λ from 0 to 1 allows for a controlled process that transfers a baseline block to pruned block as shown in Equation 2, where B_b is the block of baseline model and B_p is the pruned block of the subnet and x is the input of block. Thus all the pruned blocks of the subnet go through the same process with λ and obtain the pruned subnet smoothly.

$$o = (1 - \lambda) \cdot B_b(x) + \lambda \cdot B_p(x) \quad (2)$$

Our method controls λ transition from 0 to 1 during the first training phase, and then keeps λ constant during the second phase of training as shown in Equation 3, where K is hyper-parameter and C is current training epoch and T is total training epoch.

$$\lambda = \begin{cases} 1 - \max(0, \cos(\frac{C \cdot K}{T} \cdot \frac{\pi}{2})), & \text{if } C \leq \frac{T}{K}, \\ 1, & \text{if } C > \frac{T}{K}. \end{cases} \quad (3)$$

It is worth noting that, to reduce the error in subsequent subnet merging, it is necessary to modify the padding and stride of related convolutional layers in pruned blocks before subnet training. For example, accumulate all padding

values which are not zero forward to the first convolutional layer of the block and set the padding values of the remaining convolutional layer to zero. Also, accumulate all stride values which are not equal to one backward to the last convolutional layer of the block and set the stride value of other convolutional layers to one.

Subnet Merging. After subnet training, we obtain a subnet with some activation layers replaced with Identity layers, some LayerNorm layers replaced with BatchNorm layers with some activation layers, and BatchNorm inserted at the end of the pruned block. In this stage, we adopt reparameterization techniques to make the subnet shallower.

- $Conv + BN \rightarrow Conv$

During the inference phase of a neural network, it is possible to fuse the operations of BatchNorm layers into Convolutional (Conv) layers to accelerate model inference. We assume that the parameters of the Conv layer are denoted as ω and b , and the parameters of the BN layer are denoted as $\gamma, \sigma, \epsilon, \beta$. After merging of the Conv layer and BN layer, the parameters of the Conv layer would be modified as follows:

$$\hat{\omega} = \frac{\gamma \cdot \omega}{\sqrt{\sigma^2 + \epsilon}} \quad \hat{b} = \beta + \gamma \cdot \frac{b - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad (4)$$

- $Conv/FC + Conv/FC \rightarrow Conv/FC$

Two adjacent full-connection (FC) layers can be simply merged into a FC layer by the fusion of their weights. Sup-

pose that there are two adjacent FC layers, their weights are $W_1 \in R^{C_1 \times C_0}$ and $W_2 \in R^{C_2 \times C_1}$, and their biases are $b_1 \in R^{C_1}$ and $b_2 \in R^{C_2}$. For a given input feature $x \in R^{N \times C_0}$, the output of these two FC layers is expressed as $W_2(W_1x^T + b_1) + b_2$, which can be obtained by an equivalent FC layer whose weight is equal to W_2W_1 and bias is equal to $(W_2b_1 + b_2)$.

We will primarily introduce the fusion methods between convolutional layers. For sequential 1×1 convolutional layer fusing with $k \times k$ convolutional layer, we adopt the proposed fusion method from DBB (Ding et al. 2021a) to merge the two layers into an equivalent $k \times k$ convolutional layer. For sequential $k \times k$ convolutional layer fusing with $k \times k$, it would obtain an equivalent $(2k - 1) \times (2k - 1)$ convolutional layer with more parameters and flops compared with $k \times k$ convolutional layer. We propose a simple way to address this problem, which replaces the first $k \times k$ Conv layer with a 1×1 Conv layer, where the parameter of 1×1 Conv is taken from the central point of the $k \times k$ Conv with a retraining model. Then take the DBB fusion method to transform the two convolutional layers into an equivalent $k \times k$ convolutional layer.

- $Conv + skip\ connection(Identity/Conv) \rightarrow Conv$

RepVGG (Ding et al. 2021b) proposes a method to transform a multi-branch model into an equivalent single-path model. According to the additivity property of convolutions, for two convolutional layers with the same kernel size, they satisfy the following Equation 5:

$$Conv(W_1, x) + Conv(W_2, x) = Conv(W_1 + W_2, x) \quad (5)$$

where $Conv(W, x)$ represents the convolutional operation, W_1 and W_2 are the convolutional kernel parameters, and x is the input data. RepVGG has demonstrated that the identity and 1×1 convolutional can be equivalently transformed into a $k \times k$ convolution. Then with the property of convolutional additivity, the multi-branch convolutional layers can be merged into an equivalent convolutional layer and skip connection identity can be merged into convolutional layer too.

Depth Pruner on CNN

Applying our method on CNN models can refer to Figure 2 showing the pipeline. We should find the basic block first, and design a corresponding pruned block by reference of the pruned block in Figure 2. For activation layers in the block, we replace it with an Identity layer. For the normalization layer, which is not BatchNorm layer in block, we replace it with a BatchNorm layer, otherwise nothing needs to be done. Finally, we would insert an activation layer with a BatchNorm layer at the end of block. If an activation layer already exists in the position like ResNet34 block, only a BatchNorm layer needs be inserted after the activation layer at the end of block. After the pruned block is completed, review the supernet training, subnet searching, subnet training, and subnet merging processes. We would obtain the pruned CNN models. For plain CNN models, we can define the block which can includes two or more sequential convolutional layers.

Depth Pruner on Vision Transformer

We also apply our proposed depth pruner on vision transformer models. The vision transformer block usually has a multi-headed self-attention (MHSA) module and a MLP module that includes two full-connection layers. Particularly, we utilize DeiT (Touvron et al. 2021) as the case showing the pruning flows. As demonstrated in Figure 2, to build the Supernet, we add BN bypasses next to the LN and activation (GELU) layers of the original model and insert a GELU&BN block after the residual addition operation. After subnet searching and subnet training, we obtain the subnet, whose original LN and GELU operations of the pruned blocks are all replaced by BNs. A GELU&BN block is attached after the residual addition. Then, we merge the subnet to obtain a fast pruned model as shown in Figure 2.

Experiments

In this section, we showcase the efficacy of our depth pruner. Initially, we elucidate the experimental configurations and outline the procedure for applying the depth pruner to both CNN models and Vision Transformers. Subsequently, we compare our results with the state-of-the-art pruning methods to highlight the superiority of our approach. Finally, we perform ablation studies to elaborate on the effect of subnet searching and progressive training strategy in our method.

Datasets

All the experiments are conducted on the ImageNet-1K (Russakovsky et al. 2015). ImageNet-1K dataset is a widely used image classification dataset that spans 1000 object classes and contains 1,281,167 training images, 50,000 validation images, and 100,000 test images. We apply conventional data augmentation techniques to preprocess input images during training and scale input images to 224×224 for all experiments with reporting performance on validation dataset.

Experiments Setting on Different Models

We apply depth pruner on a series of CNN models, including ResNet34 (He et al. 2016), MobileNetV2, ConvNeXtV1 (Liu et al. 2022), and Vision Transformer (Touvron et al. 2021) to validate the efficiency of our method. We utilize four GPUs to train our model, with a total batch size of 256. In the training process, we take 10 epochs to train the supernet, except for MobileNetV2 and search optimal subnets. Then we train these subnets with the proposed progressive training strategy and complete subnet merging to obtain more efficient shallow models.

ResNet34. For ResNet34 pruning experiments, we prune 6 and 10 blocks respectively and go through the whole pruning process to obtain two pruned and shallow subnets. For subnet training, the hyper-parameters K in Equation 3 is 3, and the total training epochs is 150. At epoch 100, we change kernel size from 3×3 to 1×1 of the first convolutional layer in the pruned block. We compare our method with MetaPruning (Liu et al. 2019) and channel-wise NAS method Universally Slimmable Networks (US) (Yu and Huang 2019) to verify the pruning performance.

Models	FLOPs (G)	Acc1 (%)	Speedup
ResNet34-Baseline	3.67	73.6	1.00
US-ResNet34-0.6×	2.32	72.0	1.24
MetaPruning-0.6×	2.32	72.8	1.24
Ours-P6	2.97	73.2	1.25
Ours-P6*	2.97	73.5	1.24
US-ResNet34-0.5×	1.90	70.8	1.43
MetaPruning-0.5×	1.87	71.6	1.43
Ours-P10	2.51	71.8	1.43
Ours-P10*	2.51	72.4	1.43

Table 1: Classification performance comparisons on ImageNet. P6 and P10 indicate pruning 6 and 10 blocks, respectively, and '*' means a higher accuracy subnet with longer search time.

MobileNetV2. For MobileNetV2 pruning experiments, we adopt three pruning configurations from DepthShrinker. We skip the supernet training and subnet searching phases to obtain three same subnets. We directly train these three subnets to the end and compare the final performance with the corresponding subnet. We also use the NAS method to search three subnets with similar speedup ratios to compare with our method. For subnet training, the hyper-parameters K is 3, and the total training epochs is 450.

ConvNeXtV1. For ConvNeXtV1 pruning experiments, we set three pruning configurations which let the subnet obtain the performance around 81% top-1 accuracy on ImageNet-1k. Then, we compare these subnets with many SOTA models which include CNN and vision transformer structures to verify our method pruning performance. For subnet training, the hyper-parameters K is 4.5 and the total training epochs is 450. To achieve better speedup, we change the kernel of the depthwise conv of the pruned block from 7 to 3.

DeiT. For DeiT pruning experiments, we conduct a pruning experiment with 6 layers experiment to compare with the SOTA vision transformer pruning method. For subnet training, the hyper-parameters K is 6, and the total training epochs is 450.

Comparisons with SOTA Models

We compare the depth pruner with the state-of-the-art pruning methods under the comparable inference speed on a single AMD MI100 GPU. Following (Graham et al. 2021), we measure the average inference speedup of compressed networks with batchsize=128. In this paper, we compare the accuracy of different models with comparable speedups.

ResNet34. Table 1 compares our method with MetaPruning and NAS US methods on ResNet34. We prune 6 and 10 blocks respectively by applying our depth pruner to obtain two subnets with 1.25× and 1.43× speedup ratios, respectively. Under comparable speedup, our method surpasses the MetaPruning by 0.8% and NAS US method by 1.6% on 1.43× speedup ratio with a longer search time.

Models	FLOPs (M)	Acc1 (%)	Speedup
MBV2-1.4-Baseline	630	76.5	1.00
MetaPruning-0.5×	332	73.2	1.49
US-MBV2-1.4-0.6×	384	73.4	1.56
MBV2-1.4-DS-A	519	74.4	1.75
Ours-P6	519	74.8	1.75
US-MBV2-1.4-0.4×	286	72.2	2.04
MBV2-1.4-DS-C	492	73.1	2.16
Ours-P9	492	73.8	2.16
US-MBV2-1.4-0.3×	213	68.1	2.40
MBV2-1.4-DS-E	474	72.2	2.50
Ours-P11	474	72.5	2.50

Table 2: Classification performance comparisons with MetaPruning, NAS US and DepthShrinker on ImageNet with same network structures as the DepthShrinker.

MobileNetV2. Table 2 shows the experimental results on MobileNetV2-1.4. We adopt the same subnets as DepthShrinker, but the subnet training process is different from our progressive training strategy. We achieve 0.7% higher accuracy than MBV2-1.4-DS-C at a 2.16× speedup ratio, and some improvement compared to DepthShrinker at other speedup ratios. We also compare MetaPruning, and similar to ResNet34, we reproduce MetaPruning-0.35× with inference speeds comparable to MBV2-1.4-DS-C, while our depth pruning achieves a 2.1% higher accuracy with a higher speedup ratio.

ConvNeXtV1. Table 3 compares our accuracy with some common efficient models since there is no compression method for ConvNeXtV1. We test the speedup ratios of all networks on the AMD platform using the slowest network EfficientFormerV2-S2 in the table as a benchmark. We divide the model into levels by accuracy, and our depth pruning method achieves higher accuracy with comparable speed in different levels.

DeiT. As shown in Table 4, our method outperforms other state-of-the-art methods in both accuracy and speedup ratio. Our proposed depth pruner achieves a 1.26× speedup ratio with only a 1.9% top-1 accuracy drop. By replacing mergeable modules and applying the reparameterization technique, our proposed method can shrink the network and bring real inference acceleration.

Ablation Study

In this section, we analyze the effectiveness of subnet searching and progressive training strategy.

Models	Before FT(%)	After FT(%)
ResNet34-P10-A	57.8	71.8
ResNet34-P10-B	55.9	71.2

Table 5: Evaluating the accuracy consistency of subnets.

Models	Type	FLOPs(G)	Params(M)	Acc1(%)	Speedup
ConvNeXtV1-T-Baseline (Liu et al. 2022)	Conv	4.5	28.6	82.1	2.4
RepVGG-B1 (Ding et al. 2021b)	Conv	11.8	51.8	78.4	2.1
EfficientFormerV2-S1 (Li et al. 2022)	Hybrid	0.7	6.1	79.0	4.0
MobileOne-S4 (Vasu et al. 2023)	Conv	3.0	14.8	79.4	3.9
ConvNeXtV2-P (Woo et al. 2023)	Conv	9.1	9.1	79.7	4.0
PVT-Small (Wang et al. 2021c)	Attention	3.8	24.5	79.8	2.7
VanillaNet-9 (Chen et al. 2023)	Conv	8.6	41.4	79.9	1.1
RegNetY-12G (Radosavovic et al. 2020)	Conv	12.1	51.8	80.3	1.8
ConvNeXtV1-T-UPDP-P9	Conv	2.5	23.6	80.6	4.9
RepVGG-B3 (Ding et al. 2021b)	Conv	26.2	110.9	80.6	1.1
ConvNeXtV1-N (Woo et al. 2023)	Conv	2.5	15.6	80.8	3.6
EdgeViT-S (Pan et al. 2022)	Hybrid	1.9	11.1	81.0	3.6
Swin-T (Liu et al. 2021)	Attention	4.5	28.3	81.1	2.8
PVT-Medium (Wang et al. 2021c)	Attention	6.7	44.0	81.2	1.8
ConvNeXtV1-T-UPDP-P6	Conv	3.1	27.5	81.3	4.2
VanillaNet-11 (Chen et al. 2023)	Conv	10.3	50.0	81.1	1.1
EfficientFormerV2-S2 (Li et al. 2022)	Hybrid	1.3	12.6	81.6	1.0
PVT-Large (Wang et al. 2021c)	Attention	9.8	61.0	81.7	1.3
DeiT-B (Touvron et al. 2021)	Attention	17.5	86.0	81.8	1.5
ConvNeXtV1-T-UPDP-P3	Conv	3.8	28.3	81.9	3.3

Table 3: Performance of ConvNeXtV1 depth pruning results on ImgeNet. Speedups are tested on an AMD MI100 GPU with a batch size of 128. Adopt the slowest network in the table (EfficientFormerV2) as the baseline(1.0 speedup) for comparison.

Models	FLOPs (G)	Params (M)	Acc1 (%)	Speedup
DeiT-Tiny	1.3	5.4	72.2	1.00
SCOP*	0.8	-	68.9	-
HVT*	0.7	-	69.7	-
S ² ViTE	1.0	4.2	70.1	1.12
WD-Pruning	0.7	3.5	70.3	1.20
XPruner	0.6	-	71.1	-
Ours-P6	0.9	3.8	70.3	1.26

Table 4: DeiT depth pruning results on ImageNet. The results of S²ViTE (Tang et al. 2022) and WD-Pruning (Yu et al. 2022) refer to their paper. SCOP (Tang et al. 2020), HVT (Pan et al. 2021), and XPruner (Yu and Xiang 2023) do not publish their results about the number of parameters and speedup ratio. "*" denotes that the results come from (Yu et al. 2022).

Effectiveness of Subnet Searching. We verify the effectiveness of our ResNet34 subnet searching by comparing performance of two pruned-10-layers subnets with different accuracy before subnet finetune (FT) based on ResNet34. Table 5 shows ResNet34-P10-A with a higher accuracy before subnet finetune can achieve higher finetune accuracy, which proves the effectiveness of supernet training and subnet searching for optimal subnet with a final high performance.

Models	Direct (%)	Progressive (%)
ResNet34-P10	71.3	71.8
MBV2-1.4-P9	73.1	73.8
ConvNeXtV1-T-P3	81.6	81.9
DeiT-Tiny-P6	69.5	70.3

Table 6: Evaluating the effectiveness of progressive training.

Effectiveness of Progressive Training Strategy. Compared with hard removal of non-linear activation functions, our progressive training has a significant improvement in the accuracy of each subnetwork. As shown in Table 6, for various sub-networks, we observe that progressive training improves accuracy by 0.3%-0.8% than direct training method.

Conclusion

In this paper, we present a unified depth pruner for both efficient CNN and vision transformer models to prune models in the depth dimension. Our depth pruner includes four steps, which are supernet training, subnet searching, subnet training, and subnet merging. We propose a novel block pruning method and a progressive training strategy to utilize baseline model weights better. During subnet merging, we use reparameterization technique to make subnet become shallower and faster. We conduct our method to several CNN models and transformer models. The SOTA pruning performance demonstrates the superiority of our method. In the future, we would explore our method on more transformer models and tasks.

References

- Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.
- Bhardwaj, K.; Milosavljevic, M.; O’Neil, L.; Gope, D.; Matas, R.; Chalfin, A.; Suda, N.; Meng, L.; and Loh, D. 2022. Collapsible linear blocks for super-efficient super resolution. *MLSys*, 4: 529–547.
- Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *NeurIPS*, 33: 1877–1901.
- Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; and Han, S. 2020. Once for All: Train One Network and Specialize it for Efficient Deployment. In *ICLR*.
- Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; and Zagoruyko, S. 2020. End-to-end object detection with transformers. In *ECCV*, 213–229. Springer.
- Chen, H.; Wang, Y.; Guo, J.; and Tao, D. 2023. VanillaNet: the Power of Minimalism in Deep Learning. *arXiv preprint arXiv:2305.12972*.
- Chen, S.; and Zhao, Q. 2018. Shallowing deep networks: Layer-wise pruning based on feature representations. *TPAMI*, 41(12): 3048–3056.
- Chollet, F. 2017. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 1251–1258.
- Ding, X.; Zhang, X.; Han, J.; and Ding, G. 2021a. Diverse branch block: Building a convolution as an inception-like unit. In *CVPR*, 10886–10895.
- Ding, X.; Zhang, X.; Ma, N.; Han, J.; Ding, G.; and Sun, J. 2021b. Repvgg: Making vgg-style convnets great again. In *CVPR*, 13733–13742.
- Dong, Y.; Cordonnier, J.-B.; and Loukas, A. 2021. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *ICML*, 2793–2803. PMLR.
- Dror, A. B.; Zehngut, N.; Raviv, A.; Artyomov, E.; Vitek, R.; and Jevnisek, R. 2021. Layer folding: Neural network depth reduction using activation linearization. *arXiv preprint arXiv:2106.09309*.
- Elkerdawy, S.; Elhoushi, M.; Singh, A.; Zhang, H.; and Ray, N. 2020. To filter prune, or to layer prune, that is the question. In *ACCV*.
- Elsen, E.; Dukhan, M.; Gale, T.; and Simonyan, K. 2020. Fast sparse convnets. In *CVPR*, 14629–14638.
- Fu, Y.; Yang, H.; Yuan, J.; Li, M.; Wan, C.; Krishnamoorthi, R.; Chandra, V.; and Lin, Y. 2022. DepthShrinker: a new compression paradigm towards boosting real-hardware efficiency of compact neural networks. In *ICML*, 6849–6862. PMLR.
- Graham, B.; El-Nouby, A.; Touvron, H.; Stock, P.; Joulin, A.; Jégou, H.; and Douze, M. 2021. Levit: a vision transformer in convnet’s clothing for faster inference. In *ICCV*, 12259–12269.
- Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.
- He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.
- He, Y.; Liu, P.; Wang, Z.; Hu, Z.; and Yang, Y. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*, 4340–4349.
- Howard, A.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. 2019. Searching for mobilenetv3. In *ICCV*, 1314–1324.
- Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.
- Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.
- Li, Y.; Hu, J.; Wen, Y.; Evangelidis, G.; Salahi, K.; Wang, Y.; Tulyakov, S.; and Ren, J. 2022. Rethinking vision transformers for mobilenet size and speed. *arXiv preprint arXiv:2212.08059*.
- Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; and Guo, B. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 10012–10022.
- Liu, Z.; Mao, H.; Wu, C.-Y.; Feichtenhofer, C.; Darrell, T.; and Xie, S. 2022. A ConvNet for the 2020s. *CVPR*.
- Liu, Z.; Mu, H.; Zhang, X.; Guo, Z.; Yang, X.; Cheng, K.-T.; and Sun, J. 2019. Metapruning: Meta learning for automatic neural network channel pruning. In *ICCV*, 3296–3305.
- Michel, P.; Levy, O.; and Neubig, G. 2019. Are sixteen heads really better than one? *NeurIPS*, 32.
- Pan, J.; Bulat, A.; Tan, F.; Zhu, X.; Dudziak, L.; Li, H.; Tzimiropoulos, G.; and Martinez, B. 2022. Edgevits: Competing light-weight cnns on mobile devices with vision transformers. In *ECCV*, 294–311. Springer.
- Pan, Z.; Zhuang, B.; Liu, J.; He, H.; and Cai, J. 2021. Scalable vision transformers with hierarchical pooling. In *ICCV*, 377–386.
- Pool, J.; and Yu, C. 2021. Channel permutations for N: M sparsity. *NeurIPS*, 34: 13316–13327.
- Pope, R.; Douglas, S.; Chowdhery, A.; Devlin, J.; Bradbury, J.; Heek, J.; Xiao, K.; Agrawal, S.; and Dean, J. 2023. Efficiently scaling transformer inference. *MLSys*, 5.
- Radosavovic, I.; Kosaraju, R. P.; Girshick, R.; He, K.; and Dollár, P. 2020. Designing network design spaces. In *CVPR*, 10428–10436.
- Rao, Y.; Zhao, W.; Liu, B.; Lu, J.; Zhou, J.; and Hsieh, C.-J. 2021. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *NeurIPS*, 34: 13937–13949.
- Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *Int J Comput Vis*, 115: 211–252.
- Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520.

- Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Strudel, R.; Garcia, R.; Laptev, I.; and Schmid, C. 2021. Seg-menter: Transformer for semantic segmentation. In *ICCV*, 7262–7272.
- Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; and Le, Q. V. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2820–2828.
- Tan, M.; and Le, Q. 2021. Efficientnetv2: Smaller models and faster training. In *ICML*, 10096–10106. PMLR.
- Tang, Y.; Han, K.; Wang, Y.; Xu, C.; Guo, J.; Xu, C.; and Tao, D. 2022. Patch slimming for efficient vision transformers. In *CVPR*, 12165–12174.
- Tang, Y.; Wang, Y.; Xu, Y.; Tao, D.; XU, C.; Xu, C.; and Xu, C. 2020. SCOP: Scientific Control for Reliable Neural Network Pruning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *NeurIPS*, volume 33, 10936–10947. Curran Associates, Inc.
- Touvron, H.; Cord, M.; Douze, M.; Massa, F.; Sablayrolles, A.; and Jegou, H. 2021. Training data-efficient image transformers amp; distillation through attention. In *ICML*, volume 139, 10347–10357.
- Vasu, P. K. A.; Gabriel, J.; Zhu, J.; Tuzel, O.; and Ranjan, A. 2023. MobileOne: An Improved One Millisecond Mobile Backbone. In *CVPR*, 7907–7917.
- Wang, D.; Gong, C.; Li, M.; Liu, Q.; and Chandra, V. 2021a. Alphanet: Improved training of supernets with alpha-divergence. In *ICML*, 10760–10771. PMLR.
- Wang, D.; Li, M.; Gong, C.; and Chandra, V. 2021b. Attention-venas: Improving neural architecture search via attentive sampling. In *CVPR*, 6418–6427.
- Wang, W.; Xie, E.; Li, X.; Fan, D.-P.; Song, K.; Liang, D.; Lu, T.; Luo, P.; and Shao, L. 2021c. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *ICCV*, 568–578.
- Woo, S.; Debnath, S.; Hu, R.; Chen, X.; Liu, Z.; Kweon, I. S.; and Xie, S. 2023. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *CVPR*, 16133–16142.
- Wu, Y.; and He, K. 2018. Group normalization. In *ECCV*, 3–19.
- Xu, P.; Cao, J.; Shang, F.; Sun, W.; and Li, P. 2020. Layer pruning via fusible residual convolutional block for deep neural networks. *arXiv preprint arXiv:2011.14356*.
- Yu, F.; Huang, K.; Wang, M.; Cheng, Y.; Chu, W.; and Cui, L. 2022. Width amp; Depth Pruning for Vision Transformers. *AAAI*, 36(3): 3143–3151.
- Yu, J.; and Huang, T. S. 2019. Universally slimmable networks and improved training techniques. In *ICCV*, 1803–1811.
- Yu, J.; Jin, P.; Liu, H.; Bender, G.; Kindermans, P.-J.; Tan, M.; Huang, T.; Song, X.; Pang, R.; and Le, Q. 2020. Bignas: Scaling up neural architecture search with big single-stage models. In *ECCV*, 702–717. Springer.
- Yu, L.; and Xiang, W. 2023. X-Pruner: eXplainable Pruning for Vision Transformers. *ArXiv*, abs/2303.04935.
- Zhang, M.; and He, Y. 2020. Accelerating training of transformer-based language models with progressive layer dropping. *NeurIPS*, 33: 14011–14023.
- Zhou, Y.; Yen, G. G.; and Yi, Z. 2021. Evolutionary shallowing deep neural networks at block levels. *TNNLS*, 33(9): 4635–4647.
- Zhu, M.; Tang, Y.; and Han, K. 2021. Vision transformer pruning. *arXiv preprint arXiv:2104.08500*.