Trunk Pruning: Highly Compatible Channel Pruning for Convolutional Neural Networks Without Fine-Tuning

Nam Joon Kim, *Graduate Student Member, IEEE*, and Hyun Kim, *Senior Member, IEEE*

***Abstract*—Channel pruning can efficiently reduce the computation and memory footprint within a reasonable accuracy drop by removing unnecessary channels from convolutional neural networks (CNNs). Among the various channel pruning approaches, sparsity training is the most popular because of its convenient implementation and end-to-end training. It automatically identifies the optimal network structures by applying regularization to parameters. Although this sparsity training has achieved a remarkable performance in terms of the trade-off between accuracy and network size reduction, it needs to be accompanied by a time-consuming fine-tuning process. Moreover, although activation functions with high performance are being continuously developed, the existing sparsity training does not display remarkable scalability for these new activation functions. To address these problems, this study proposes a novel pruning method, *trunk pruning*, which can produce a compact network by minimizing the accuracy drop during inference even without the fine-tuning process. In the proposed method, one kernel of the next convolutional layer absorbs all the information of the kernels to be pruned, considering the effects of the batch normalization (BN) shift parameters remaining after the sparsity training. Therefore, it is possible to eliminate the fine-tuning process because trunk pruning can effectively reproduce the output of the unpruned network after the sparsity training by removing the pruning loss. Furthermore, because trunk pruning is a technique that can effectively control only the shift parameters of the BN in the CONV layer, it has the significant advantage of being compatible with all BN-based sparsity training schemes and can address various activation functions.**

***Index Terms*—Convolutional Neural Network (CNN), Pruning, Regularization, Fine-Tuning.**

I. Introduction

In RECENT years, convolutional neural networks (CNNs) have been studied by several researchers. They have achieved the best performance on a variety of computer vision tasks, including image classification [1], [2], [3], [4], [5], [6] object detection [7], [8], [9], [10], [11], and segmentation [12], [13]. However, the performance improvement is accompanied by high computational costs, memory footprints, and power consumption [14], [15], [16], [17]. This hinders the effective deployment of CNNs on resource-constrained mobile/edge devices. Therefore, network compression and acceleration studies as well as performance improvement studies have been actively conducted for the practical use of CNNs in real-world applications [18], [19], [20], [21], [22]. Among these studies, network pruning can significantly reduce the model size and computation while maintaining reasonable accuracy by removing unnecessary filters from the convolutional (CONV) layers [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33].

State-of-the-art (SOTA) studies on various channel pruning approaches have solved the problem of the significant accuracy drop caused by the suppression of an excessive number of channels in the network, which is a disadvantage of channel pruning. Among these approaches, sparsity training [24], [26], [34] has received significant attention owing to its simple implementation and high performance. It automatically identifies unnecessary channels during training by applying regularization (e.g., ℓ-1 or ℓ-2). The following three processes generally characterize the channel pruning method based on sparsity training: 1) Sparsity training: In general, the channel redundancy is generated in the corresponding channels or filters by imposing sparsity regularization on the batch normalization (BN) [35] scaling factors. 2) Pruning: The filters and output channels with scaling factors lower than the predefined threshold, and the corresponding kernels of the next CONV layer are removed. 3) Fine-tuning: After the pruning, the pruned network is fine-tuned to recover the lost performance.

The accuracy loss after the pruning occurs because, during the sparsity training, the BN scaling parameter is close to zero, but the remaining shift parameter is not considered. As a result, a distortion is generated during the inference of the pruned network. This, in turn, causes the network to make incorrect predictions. Moreover, the pruned networks with a reduced representational capacity can be easily trapped into bad local minima [31], [36]. Therefore, most existing studies consider fine-tuning to compensate for the accuracy drop. However, this fine-tuning process is time-consuming, and the use of GPUs for training CNNs has increased tremendously in recent years [37]. This causes severe environmental problems, such as a substantial CO2 emission. In

Manuscript received 18 November 2022; revised 22 February 2023, 9 May 2023, and 26 October 2023; accepted 22 November 2023. Date of publication 30 November 2023; date of current version 21 March 2024. This work was supported by the MSIT (Ministry of Science and ICT), Korea through ITRC (Information Technology Research Center) Program under Grant IITP-2023-RS-2022-00156295, supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation). The Associate Editor coordinating the review of this manuscript and approving it for publication was Prof. Xiaochun Cao. *(Corresponding author: Hyun Kim.)*

The authors are with the Department of Electrical and Information Engineering, The Research Center for Electrical and Information Technology, Seoul National University of Science and Technology, Seoul 01811, South Korea (e-mail: rlarla2626@seoultech.ac.kr; hyunkim@seoultech.ac.kr).

Digital Object Identifier 10.1109/TMM.2023.3338052

TABLE I
PERFORMANCE COMPARISON BETWEEN THE PRUNED RESNET-56 AND 164 USING THREE SPARSITY TRAINING METHODS ON THE CIFAR-10 DATASET

Network	Sparsity Training	Pruning Method	Baseline (%)	Pruned Acc. before Fine-Tuning (%)	Pruned Acc. after Fine-Tuning (%)	Acc. Drop before Fine-Tuning(%)	FLOPs reduction (%)
Res56	Slimming	Trunk	93.57	91.41	91.21	2.16	76.30
	Slimming	Conventional	93.57	83.15	91.11	10.42	77.06
	GSM	Trunk	93.57	93.04	92,91	0.53	58.86
	GSM	Conventional	93.57	88.98	92.49	4.59	61.34
	GBN	Trunk	93.57	92.70	92.25	0.87	70.58
	GBN	Conventional	93.57	90.87	92.36	2.70	71.28
Res164	Slimming	Trunk	94.28	93.68	93.08	0.87	78.55
	Slimming	Conventional	94.28	88.85	93.48	5.43	79.32
	GSM	Trunk	94.28	94.03	93.79	0.25	70.95
	GSM	Conventional	94.28	91.57	93.83	2.71	72.31
	GBN	Trunk	94.28	93.93	93.60	0.35	75.75
	GBN	Conventional	94.28	90.99	93.72	3.29	76.66

addition, the existing methods cannot ensure the scalability of several activation functions.

To solve these problems, we propose a novel pruning method, trunk pruning, which can eliminate the time-consuming fine-tuning process and has high compatibility and scalability with BN-based sparsity training. Specifically, when the BN scaling parameters are close to zero after the sparsity training, the output before the pruning can be entirely reproduced by including the remaining kernel weights into a kernel (trunk) in the following CONV layer, considering the shift parameters. Therefore, the fine-tuning process can be eliminated because the pruned network can maintain the same output as the non-pruned network. The main contributions of our study are summarized below:

- Exclusion of the fine-tuning: Trunk pruning, which enables the pruned network to effectively reproduce the output of the unpruned network after the sparsity training, displays a negligible accuracy drop after the pruning. Consequently, it eliminates the fine-tuning process, which is time-consuming and requires significant hardware resources. Specifically, several pruned ResNets [1] and VGG [38] achieved SOTA performance in the CIFAR-10 [39] and ImageNet ILSVRC-12 [40] datasets, compared with the existing channel pruning methods (see Tables II and IV).
- Compatibility with the existing sparsity training methods: The proposed trunk pruning effectively controls the BN shift parameters. Hence, it can be applied to any BN-based sparsity training. The compatibility of the method is highlighted by the results obtained using the trunk pruning for two channel-pruning methods and a weight-pruning method listed in Table I.
- Compatibility with various activation functions: Since the ReLU considers negative values as zero, it is relatively insensitive to an accuracy drop owing to the shift parameter. However, the activation functions with negative regions, such as Mish [41] and Swish [42], are significantly more sensitive to the accuracy drop due to the shift parameter. Therefore, trunk pruning is significantly more effective for activation functions with negative values. These results are presented in Tables VI and VII.

TABLE II
PERFORMANCE COMPARISON BETWEEN THE CHANNEL PRUNING METHODS ON THE CIFAR-10 DATASET

Network	Method	Fine-Tuning?	Baseline (%)	Pruned Acc.(%)	Acc. Drop(%)	FLOPs reduction(%)
Res56	**Ours**	×	**93.57**	**93.58**	**-0.01**	**50**
	Variational [29]	×	93.04	92.26	0.78	20.3
	SCP [31]	×	93.69	93.23	0.46	51.5
	DSA [52]	✓	93.12	93.08	0.04	29.3
	NISP [50]	✓	-	-	0.03	43.61
	CCP [53]	✓	93.5	93.46	0.04	47
	Polarization [46]	✓	93.8	93.83	-0.03	47
	Hrank [54]	✓	93.26	93.17	0.09	50
	CP [23]	✓	92.8	91.8	1.00	50
	MFP [55]	✓	93.59	93.56	0.03	52.6
	LFPC [30]	✓	93.59	93.24	0.35	52.9
Res164	**Ours**	×	**94.28**	**94.22**	**0.06**	**67.42**
	Variational [29]	×	93.58	93.16	0.42	49.08
	C-SGD [27]	×	94.83	94.75	0.08	60.91
	ResRep [56]	×	94.99	94.57	0.42	65.00
Vgg19	**Ours**	×	**92.98**	**93.14**	**-0.16**	**73.4**
	ResRep [56]	×	93.48	93.16	0.32	59.7
	SCP [31]	×	93.84	93.82	0.02	74.06

The bold values represent the key results of our experiments.

TABLE III
PERFORMANCE COMPARISON BETWEEN THE CHANNEL PRUNING METHODS USING RESNET-56 ON THE CIFAR-100 DATASET

Network	Method	Fine-Tuning?	Baseline (%)	Pruned Acc.(%)	Acc. Drop(%)	FLOPs reduction(%)
Res56	**Ours**	×	**70.97**	**69.9**	**1.07**	**53.41**
	FPGM [25]	×	71.41	69.66	1.75	52.6
	SFP [57]	×	71.4	68.79	2.61	52.6
	ResRep [56]	×	72.35	71.14	1.21	54.86
	QSFM [58]	✓	70.62	68.36	2.26	53.87

The bold values represent the key results of our experiments.

TABLE IV
PERFORMANCE COMPARISON WITH THE CHANNEL PRUNING METHODS ON THE IMAGENET ILSVRC-12 DATASET

Network	Method	Fine-Tuning?	Baseline (%)	Pruned Acc.(%)	Acc. Drop(%)	FLOPs reduction(%)
Res50	**Ours**	×	**76.1**	**76.44**	**-0.34**	**39.29**
	Ours	×	**76.1**	**75.62**	**0.48**	**60.85**
	ResRep [56]	×	76.13	75.93	0.2	35.01
	SSS [59]	×	76.12	71.82	4.30	43
	FPGM [25]	×	76.15	74.13	2.02	53.5
	SCP [31]	×	75.89	74.2	1.69	54.3
	ResRep [56]	×	76.15	75.3	0.85	62.1
	Taylor-FO [28]	✓	76.18	75.48	0.7	34.96
	DSA [52]	✓	76.02	75.1	0.92	40
	EPruner [60]	✓	76.01	74.95	1.06	42.7
	DCP [61]	✓	76.01	75.15	0.86	52.41
	MFP [55]	✓	76.15	74.86	1.29	53.5
	Polarization [46]	✓	76.15	75.63	0.52	54
	GBN-50 [34]	✓	75.85	75.18	0.67	55.06
	Hrank [54]	✓	76.15	71.98	4.17	62.1
Res101	**Ours**	×	**77.16**	**76.55**	**0.61**	**54.41**
	Rethinking [26]	✓	76.4	74.56	1.84	52.69
	Taylor-FO [28]	✓	77.37	75.95	1.42	63.46

The bold values represent the key results of our experiments.

TABLE V
PERFORMANCE COMPARISON WITH CHANNEL PRUNING METHODS USING MOBILENET-V1 ON THE IMAGENET ILSVRC-12 DATASET

Network	Method	Fine-Tuning?	Baseline (%)	Pruned Acc.(%)	Acc. Drop(%)	FLOPs
MobileNet-V1	**Trunk**	×	**71.75**	**70.61**	**1.14**	**328M**
	Conventional	×	71.75	0.21	71.54	325M
	Trunk	×	**71.75**	**69.73**	**2.02**	**280M**
	Conventional	×	71.75	0.11	71.64	277M
	NetAdapt [62]	✓	-	69.1	-	284M

The bold values represent the key results of our experiments.

TABLE VI
THE PRUNING RESULTS AFTER THE GBN-BASED SPARSITY TRAINING ON VARIOUS NETWORKS USING VARIOUS ACTIVATION FUNCTIONS WITH NEGATIVE VALUES

Network/ Dataset	Pruning Method	Activation	Fine-Tuning?	Baseline (%)	Pruned Acc.(%)	Acc. Drop(%)	FLOPs reduction (%)
Res56/ CIFAR-10	Trunk	Mish	×	93.77	93.75	0.02	53.1
	Conventional	Mish	×	93.77	15.03	78.74	53.9
	Trunk	Swish	×	93.97	93.57	0.4	49.93
	Conventional	Swish	×	93.97	8.51	85.45	50.53
	Trunk	LeakyReLU	×	93.51	93.04	0.47	52.42
	Conventional	LeakyReLU	×	93.51	75.03	18.48	53.04
Res110/ CIFAR-10	Trunk	Mish	×	94.31	94.28	0.03	64.71
	Conventional	Mish	×	94.31	26.05	68.26	65.14
	Trunk	Swish	×	93.92	94.19	-0.27	64.97
	Conventional	Swish	×	93.92	29.16	64.76	65.99
Res50/ ImageNet	Trunk	Mish	×	77.55	76.41	1.14	50.9
	Conventional	Mish	×	77.55	0.12	77.43	51.5
	Trunk	Swish	×	77.51	76.7	0.81	51.8
	Conventional	Swish	×	77.51	0.01	77.5	52.4
	Trunk	LeakyReLU	×	77.66	75.81	1.85	48.8
	Conventional	LeakyReLU	×	77.66	15.74	61.92	49.4

The remainder of this article is organized as follows. Section II introduces existing studies related to pruning. Section III provides a detailed description of the proposed trunk pruning. Section IV presents the experimental results and an analysis of the proposed method. Finally, Section V concludes this article.

II. RELATED WORKS

A. Weight Pruning

Through several heuristics or optimization processes, the weight pruning results in an unstructured sparsity in the network. Therefore, it is generally used with quantization [18]. The OBD [43] determines the importance of the connection using the Hessian matrix of the loss function and removes those with low importance. The DSD [44] introduces a dense-sparse-dense training framework that restores the connection after pruning for regularizing deep neural networks (DNNs). The GSM [45] evaluates the importance of the weights using the first-order Taylor series and then divides the existing momentum-based stochastic gradient descent (SGD) update rules into two types: passive and active updates. The modified update rule of the GSM eliminates the fine-tuning process because it can completely zero out the weights that are unimportant. However, weight pruning cannot improve the inference speed of DNNs without using customized software (SW) or hardware (HW).

B. Filter Pruning w/ Extra Fine-Tuning

The channel pruning can achieve structured sparsity by removing the filters/channels of the CNNs. This structured sparsity is optimized for the basic linear algebra subprograms (BLAS) library. Consequently, it is possible to accelerate the training and inference speeds without customized SW/HW support. Sparsity training is characterized by convenient implementation and end-to-end training. It is an extremely efficient and popular approach among the many channel pruning methods. Slimming [24] obtains a structured sparsity by imposing the ℓ-1 regularization on the BN scaling parameters during the sparsity training and then removing the filters below a predefined threshold in the pruning step. Subsequently, fine-tuning is performed to restore the performance loss to a reasonable level. The GBN [34] uses the Taylor expansion to estimate the importance of the channels. The tick-tock framework of the GBN [34] imposes a sparsity penalty (tick) only on unimportant scaling parameters and fine-tunes the network after the pruning process (tock). Polarization [46] proposes polarization regularization to separate the filters to be removed and those to be preserved. This demonstrates the effectiveness of the proposed regularization method, both theoretically and experimentally. However, these conventional channel pruning methods remove the channels while ignoring the influence of the BN shift parameter. This results in a severe performance loss because the network can propagate distorted outputs during forward propagation. In particular, these conventional pruning methods can cause a significant pruning error in the networks that use activation functions with a negative value. Therefore, time-consuming fine-tuning is essential to recovering the lost performance. However, because of their smaller representational capacity, fine-tuning causes the pruned networks easier to get trapped in bad local minima [27], [31].

Neural architecture search (NAS)-based channel pruning is another method that requires fine-tuning to identify an optimal neural architecture that can maintain a high level of accuracy under the required inference and computation budgets. Metapruning [47] first trains an auxiliary network called PruningNet. Then, PruningNet receives network encoding vectors as input and generates the weights for the pruned network. Finally, the best-pruned network is fine-tuned. In [48], the objective was to determine the optimal number of channels for each layer, unlike the existing channel pruning method that selects important channels, and a channel pruning method based on the artificial bee colony algorithm was proposed to identify the optimal network structure. However, these NAS-based channel pruning methods require the use of many GPUs owing to the substantial search cost to identify the optimal network structure.

C. Filter Pruning w/o Extra Fine-Tuning

Several studies have been conducted to solve the problem of time-consuming fine-tuning. The Variational [29] eliminates the fine-tuning process by reformulating the BN affine function

TABLE VII
PERFORMANCE COMPARISON OF PRUNED RESNET-56 WITH EXISTING METHODS ON THE CIFAR-10 DATASET USING TWO ACTIVATION FUNCTIONS

Method	Activation	Fine-Tuning?	Baseline (%)	Pruned Acc. (%)	Acc. Drop (%)	FLOPs reduction (%)	Training epochs	Approximate training time
Ours	Mish	×	**93.77**	**93.75**	**0.02**	**53.1**	**300+300**	**6 hours**
ResRep [56]	Mish	×	93.97	93.09	0.88	53.3	300+300	6.5 hours
C-SGD [27]	Mish	×	94.06	93.82	0.24	52.7	300+300	7.5 hours
AOFP [63]	Mish	✓	94.07	93.27	0.8	55.0	300+150+150	8 hours
Ours	Swish	×	**93.97**	**93.57**	**0.4**	**49.9**	**300+300**	**6 hours**
ResRep [56]	Swish	×	93.99	93.21	0.78	51.5	300+300	6.5 hours
C-SGD [27]	Swish	×	93.74	92.33	1.41	52.7	300+150	6 hours
C-SGD [27]	Swish	×	93.74	93.34	0.4	49.8	300+300	8 hours
AOFP [63]	Swish	✓	94.38	93.61	0.77	54.4	300+150+150	9.5 hours

The bold values represent the key results of our experiments.

so that only the scaling parameter is considered to estimate the importance of the channel. In addition, unlike the existing deterministic method, the importance of the channel is evaluated probabilistically to ensure stable pruning. However, this reformulation method modifies the flow and value of the gradients in the network and cannot ensure optimal performance. The SCP [31] optimizes the network with differentiable masks and the existing parameters. In addition, it is feasible to prune the network without fine-tuning safely because the importance of the channel is evaluated probabilistically by considering the BN and ReLU operations in conjunction. However, the SCP [31] is a pruning method that targets only ReLU. Consequently, the accuracy degradation is highly significant in the activation functions other than ReLU. The C-SGD [27] proposes a centripetal SGD that makes the filters (including the BN parameters) gradually become more similar and eventually identical. The pruning process does not damage the network because all the parameters are considered during the C-SGD training process. Therefore, fine-tuning is unnecessary. However, the C-SGD [27] cannot achieve optimal performance because it has a predefined pruning ratio (such as a predefined network structure). These predefined channel pruning methods are incapable of dealing with a variety of scenarios. Therefore, it is infeasible to achieve the best performance [34].

III. PROPOSED METHOD

A. Motivation and Overview of This Work

To address the problems presented in the existing studies, we propose trunk pruning which displays good compatibility with sparse training. We start by selecting the best-performing sparsity training method. It should be noted that the proposed method is compatible with any sparsity training method using BN, wherein techniques with better performance are being studied continuously. That is, the application of trunk pruning based on the sparsity training method with better performance results in a significantly lower accuracy loss. In general, when the BN scaling parameters are close to zero after the sparse training, the input channels of the next CONV layer may become constant because they are influenced by the remaining shift parameters. The existing channel pruning methods that use sparsity training neglect these constant channels. Meanwhile, the proposed trunk pruning considers these by absorbing all the constant channels of the kernel to be removed in one of the following CONV kernels (named the trunk filter). As a result, the proposed method can completely reproduce the model after sparsity training (i.e., not the baseline model) by preserving the influence of the remained shift parameters after sparsity training. Because this process can compensate for the losses caused by the pruning process, fine-tuning can be eliminated while maintaining the advantages of sparsity training. In addition, recently proposed CNNs with SOTA performance use activation functions with negative values (not ReLU). The trunk pruning can be applied to any activation function because it uses the value passed through the activation function. Please note that the term "Trunk" refers to the main trunk that supports the branches of a tree. We intend to express the concept of including unnecessary branches remaining on this one main trunk. Accordingly, in trunk pruning, the remaining filters /channels are absorbed into one trunk and removed.

B. Preliminary

In this subsection, we define the symbols and notations used to describe the proposed pruning method. Here, i is the CONV layer index, and $\boldsymbol{M}^{(i)} \in \mathbb{R}^{w_i \times h_i \times c_i}$ is the $w_i \times h_i$ output feature map with c_i channels. For example, the output feature map of the j-th channel in the i-th CONV layer is $\boldsymbol{M}^{(i)}_{:,:,j}$. In the i-th CONV layer, $\boldsymbol{K}^{(i)} \in \mathbb{R}^{k \times k \times c_{i-1} \times c_i}$ represents c_i CONV layer filters with a kernel size of $k \times k$. We use $\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)}, \boldsymbol{\gamma}^{(i)}, \boldsymbol{\beta}^{(i)} \in \mathbb{R}^{c_i}$ to indicate the BN parameters [35]. BN performs normalization through the mean and variance for each batch in the training process. It enables faster convergence and stabilization of the network training and is used in most advanced CNNs. The BN process can be expressed as follows:

$$x_{BN_{out}} = \boldsymbol{\gamma^{(i)}} \cdot \frac{x_{BN_{in}} - \boldsymbol{\mu^{(i)}}}{\boldsymbol{\sigma^{(i)}}} + \boldsymbol{\beta^{(i)}} \tag{1}$$

where $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$ are the accumulated mean and standard deviation, respectively, and are used to normalize each channel. $\boldsymbol{\gamma}^{(i)}$ and $\boldsymbol{\beta}^{(i)}$ are the learnable scaling and shifting parameters, respectively, for the affine transformation. Furthermore, $x_{BN_{in}}$ and $x_{BN_{out}}$ are the BN input and output, respectively. Using $*$ to represent the CONV operation, the output feature map $\boldsymbol{M}^{(i)}_{:,:,:}$

with BN is expressed as follows:

$$\boldsymbol{M}^{(i)}_{:,:,:} = Act(\boldsymbol{\gamma}^{(i)} \cdot \frac{\Sigma_{k=1}^{c_{i-1}}(\boldsymbol{M}^{(i-1)}_{:,:,k} * \boldsymbol{K}^{(i)}_{:,:,k,:}) - \boldsymbol{\mu}^{(i)}}{\boldsymbol{\sigma}^{(i)}} + \boldsymbol{\beta}^{(i)}) \tag{2}$$

where Act denotes the activation function following CONV-BN. Since the BN layer has channel-wise scaling/shifting parameters, scaling parameter $\boldsymbol{\gamma}^{(i)}$ gradually approaches zero during sparsity training for channel-wise pruning [24].

C. Trunk Pruning

As mentioned earlier, the proposed trunk pruning is aimed at ensuring that the pruned network reproduces the output of the unpruned network, thereby eliminating the need for the fine-tuning process. That is, we aim to satisfy the following relationship:

$$\boldsymbol{M}^{(i+1)}_{unpruned_{:,:,:}} \approx \boldsymbol{M}^{(i+1)}_{pruned_{:,:,:}} \tag{3}$$

where $\boldsymbol{M}^{(i+1)}_{unpruned}$ and $\boldsymbol{M}^{(i+1)}_{pruned}$ are the output feature maps of the unpruned and pruned networks, respectively, of the $(i+1)$-th layer. Let the filter index set of the layer i be $F^{(i)}$ (e.g., $F^{(2)} = \{1, 2, 3, 4\}$ if the second layer has four filters), the filter index set (such as the filter set that is the target of the pruning after the sparsity training) with scaling parameters below the threshold be $P^{(i)} = \{j \in \boldsymbol{F}^{(i)} || \boldsymbol{\gamma}^{(i)}_j | < threshold\}$, and the opposite filter index set be $U^{(i)}$. This threshold is an adjustable parameter. A large network-size reduction can generally be achieved at a high threshold, although a large accuracy drop may occur accordingly. However, there is no significant difference in performance. Therefore, we empirically fix the threshold at 0.001. The output channels corresponding to $P^{(i)}$ are $w_i \times h_i$ channels filled with a constant value $\boldsymbol{\beta}^{(i)}$ because all the scaling factors in the output channels corresponding to $P^{(i)}$ are zero. The constant channels are maintained after applying the activation function. For example, the output feature map can be represented by $\boldsymbol{M}^{(i)}_{:,:,j} = Act(\boldsymbol{\beta}^{(i)}_j) \cdot \mathbf{1}^{w_i \times h_i}$ when the j-th channel is a constant channel. Here, $\mathbf{1}^{w_i \times h_i}$ is a $w_i \times h_i$ matrix filled with ones. The conventional pruning methods remove this constant channel, $\boldsymbol{M}^{(i)}_{:,:,j}$. The neglection of the constant channel influence can result in distortion and performance degradation in the network. However, this distortion can be eliminated effectively as follows.

Let the output of the $(i+1)$-th layer immediately after the CONV operation before BN be $\boldsymbol{CO}^{(i+1)}_{:,:,:}$. After sparsity training, due to the zero scaling parameters and remaining shift parameters, we can divide the output into two parts (i.e., constant channels and non-constant channels). Consequently, $\boldsymbol{CO}^{(i+1)}_{:,:,:}$ can be expressed as follows:

$$\boldsymbol{CO}^{(i+1)}_{:,:,:} = \underbrace{\sum_{k \in U^{(i)}} (\boldsymbol{M}^{(i)}_{:,:,k} * \boldsymbol{K}^{(i+1)}_{:,:,k,:})}_{Not\ constant\ channel} + \underbrace{\sum_{k \in P^{(i)}} (\boldsymbol{M}^{(i)}_{:,:,k} * \boldsymbol{K}^{(i+1)}_{:,:,k,:})}_{Constant\ channel} \tag{4}$$

where $\boldsymbol{M}^{(i)}_{:,:,k} = Act(\boldsymbol{\beta}^{(i)}_k) \cdot \mathbf{1}^{w_i \times h_i}$ in the constant channel and $Act(\boldsymbol{\beta}^{(i)}_k)$ is a per-channel scalar. We can randomly select one from $P^{(i)}$ and define it as the trunk for the trimming kernels as with C-SGD [27]. However, it should be noted that C-SGD performs the k-means algorithm before C-SGD training to make the filters in each cluster have the same value, then removes the remaining filters, leaving only one filter in each cluster. In contrast, our method does not include any process of training filters to have the same values. Trunk pruning is a method of absorbing shift parameters in filters to be removed, determined by various BN-based sparsity methods, into an additional trunk filter to preserve them, and therefore does not involve any process of matching filter values. Based on (4), we obtain the following using the inverse of the distributive property on the constant channel part in (4):

$$\sum_{k \in P^{(i)}} (\boldsymbol{M}^{(i)}_{:,:,k} * \boldsymbol{K}^{(i+1)}_{:,:,k,:}) \approx (Act(\boldsymbol{\beta}^{(i)}_{Trunk}) \cdot \mathbf{1}^{w_i \times h_i}) * \left(\mathbf{K}^{(i+1)}_{:,:,Trunk,:} + \sum_{\substack{k \in P^{(i)} \wedge \\ k \neq Trunk}} \left(\frac{Act(\boldsymbol{\beta}^{(i)}_k)}{Act(\boldsymbol{\beta}^{(i)}_{Trunk})} \cdot \boldsymbol{K}^{(i+1)}_{:,:,k,:} \right) \right) \tag{5}$$

where $Act(\boldsymbol{\beta}^{(i)}_{Trunk}) \cdot \mathbf{1}^{w_i \times h_i}$ serves as the new input channel for the subsequent CONV operation, denoted by the first term on the right side. Furthermore, the second term on the right side indicates the kernel weights for the subsequent CONV operation. The kernel corresponding to the trunk is updated as (6), whereas the remaining kernels, filters, and channels associated with the constant channel are removed [23].

$$\boldsymbol{K}^{(i+1)}_{new_{:,:,Trunk,:}} \approx \boldsymbol{K}^{(i+1)}_{:,:,Trunk,:} + \sum_{\substack{k \in P^{(i)} \wedge \\ k \neq Trunk}} \left(\frac{Act(\boldsymbol{\beta}^{(i)}_k)}{Act(\boldsymbol{\beta}^{(i)}_{Trunk})} \cdot \boldsymbol{K}^{(i+1)}_{:,:,k,:} \right) \tag{6}$$

It is noteworthy that the proposed trunk pruning essentially requires that the shift parameters of the filters to be removed must be absorbed into one of the constant channels whose scaling parameter of the BN is zero after sparsity training (=a channel that is about to be deleted). If pruning is performed after setting a channel whose scaling parameter is not zero (=the channels that are going to be preserved) to the trunk, the inverse of the distributive property cannot be applied in (5), and eventually, in the trunk kernel calculation in (6), large approximation errors occur. The updated $\boldsymbol{K}^{(i+1)}_{new_{:,:,Trunk,:}}$ contains all the information of the removed kernels. Therefore, (5) can be modified as follows:

$$\sum_{k \in P^{(i)}} (\boldsymbol{M}^{(i)}_{:,:,k} * \boldsymbol{K}^{(i+1}_{:,:,k,:}) \approx \boldsymbol{M}^{(i)}_{:,:,Trunk} * \boldsymbol{K}^{(i+1)}_{new_{:,:,Trunk,:}} \tag{7}$$

where $\boldsymbol{M}^{(i)}_{:,:,Trunk} = Act(\boldsymbol{\beta}^{(i)}_{Trunk}) \cdot \mathbf{1}^{w_i \times h_i}$. Therefore, the output $\boldsymbol{CO}^{(i+1)}_{:,:,:}$ of the pruned network before the BN is expressed as follows:

$$\boldsymbol{CO}^{(i+1)}_{:,:,:} \approx \sum_{k \in U^{(i)}} (\boldsymbol{M}^{(i)}_{:,:,k} * \boldsymbol{K}^{(i+1}_{:,:,k,:}) + \boldsymbol{M}^{(i)}_{:,:,Trunk} * \boldsymbol{K}^{(i+1)}_{new_{:,:,Trunk,:}} \tag{8}$$

Algorithm 1: Trunk Pruning Algorithm.

Input: Model after sparsity training $\mathcal{M}$, Scaling parameter γ, Shift parameter β, # of Layers n, # of Filters m, Threshold $\mathcal{T}$
Output: The pruned model $\mathcal{M}'$

1: **for** $i \leftarrow 1$ to n **do**
2: **for** $j \leftarrow 1$ to m **do**
3: **if** $|\gamma^{(j)}| < \mathcal{T}$ and $\beta^{(j)} > 0$ **then**
4: Select one Trunk
5: **end if**
6: **end for**
7: **end for**
8: **for** $i \leftarrow 1$ to n **do**
9: **for** $j \leftarrow 1$ to m **do**
10: **if** $|\gamma^{(j)}| < \mathcal{T}$ and $\beta^{(j)} > 0$ **then**
11: TrunkPrune() via (6) in Section III-C
12: **else if** $|\gamma^{(j)}| < \mathcal{T}$ **then**
13: Prune() // Zeroing out filters
14: **end if**
15: **end for**
16: **end for**
17: **Return** $\mathcal{M}'$

where the first term on the right is the not constant channel and the second term is the convolution operation using the trunk. Finally, we obtain the output of the pruned network after the BN as follows:

$$\boldsymbol{M}^{(i+1)}_{pruned_{:,:,:}} \approx \boldsymbol{\gamma}^{(i+1)} \cdot \frac{\boldsymbol{CO}^{(i+1)}_{:,:,:} - \boldsymbol{\mu}^{(i+1)}}{\boldsymbol{\sigma}^{(i+1)}} + \boldsymbol{\beta}^{(i+1)} \quad (9)$$

This pruning process has little impact on the network because no approximation is used other than treating BN scaling factors less than the near-zero threshold (<0.001 in our environment) as zero. That is, no loss occurs during the pruning process because a pruned network almost completely reproduces the output of an unpruned network after the sparsity training. Therefore, the fine-tuning process can be eliminated. Although the process of trunk pruning using (6) consists of multiplication, division, and addition operations on the number of total layers $\times$ constant channels $\times$ kernel size, these additional operations occupy only 0.005% of the total training time. In detail, the proposed trunk pruning consumes approximately twice the time consumed by conventional pruning owing to the additional absorption into the trunk, but this pruning time is significantly short considering the entire training process including pruning.

Fig. 1 shows a practical example of the proposed trunk pruning using two consecutive CONV layers. Let the fourth, fifth, seventh, and eighth scaling parameters $\boldsymbol{\gamma}$ of the i-th layer be below the threshold (such as $P^{(i)} = \{4, 5, 7, 8\}$). We arbitrarily set the seventh filter (such as $\boldsymbol{K}^{(i)}_{:,:,:,7}$, $\boldsymbol{M}^{(i)}_{:,:,7}$, $\boldsymbol{K}^{(i+1)}_{:,:,7,:}$) of layer i to the trunk. We absorb the fourth, fifth, and eighth kernels into the seventh trunk kernel through (6) to trim the network (see Fig. 1). As a result, the seventh kernel becomes a new kernel that includes the shift parameters of the fourth, fifth, and eighth kernels, and the corresponding filters, channels, and kernels of the fourth, fifth, and eighth kernels are completely removed. Unlike the conventional pruning methods, this method leaves an additional output channel per layer corresponding to the trunk. Therefore, the proposed method causes a marginal loss in pruning ratio compared with the conventional pruning method. However, this addition is negligible because the number of channels removed from a layer of the DNNs is significantly large. Moreover, owing to the characteristics of the layer, there are cases where trunk filters are not created. Consequently, the influence of these trunks can be observed to be more insignificant. This also implies that the number of trunks and the number of layers do not necessarily match.

The overall process for generating a pruned model using trunk pruning is summarized in Algorithm 1. First, we generate a sparse model (i.e., $\mathcal{M}$) through sparsity training. Subsequently, the trunk is selected from among the filters that satisfy both the conditions $|\gamma^{(j)}| < \mathcal{T}$ and $\beta^{(j)} > 0$ for each layer (Lines 1–7). Based on the selected trunk, trunk pruning is applied to the filters/channels/kernels of the next layer that satisfy both $|\gamma^{(j)}| < \mathcal{T}$ and $\beta^{(j)} > 0$ (Lines 10–11). Then, conventional pruning is applied to filters that satisfy only $|\gamma^{(i)}| < \mathcal{T}$ (Lines 12–13). We can generate the pruned model $\mathcal{M}'$ through this process.

IV. Experiments

A. Experimental Environments

We used the CIFAR-10 [39] and ImageNet ILSVRC-2012 [40] datasets for the evaluation. These are commonly used to compare pruning performance. The CIFAR-10 dataset contains 50,000 training images (32 $\times$ 32) and 10,000 test images for 10 classes. The baseline model is trained with a batch size of 64 for 300 epochs using SGD from scratch. The initial learning rate is set to 0.1 and is divided by 10 at 50% and 75% of the total training steps. We used a Nesterov momentum of 0.9 and a weight decay of 10^{-4} for fast convergence and regularization of the model. The ImageNet ILSVRC-2012 dataset is a large-scale dataset containing 1.28 million training images and 50,000 test images of 1,000 classes. The baseline model is trained with a batch size of 128 for 80 epochs using SGD from scratch. The initial learning rate is also set to 0.1 and is divided by 10 at 50% and 75% of the total training steps. The experiments are conducted with darknet [8] on NVIDIA Geforce RTX 2080 Ti GPUs.

B. Compatibility With Various Sparsity Training Methods

In this subsection, we show the compatibility of the proposed trunk pruning using two channel-pruning schemes (Slimming [24] and GBN [34]) and a weight-pruning scheme (GSM [45]) as sparsity training methods. In particular, GSM [45], the weight pruning method, used momentum-SGD to remove unnecessary weights from the filter. Therefore, we apply the sparsity training method of the GSM in units of BN scaling factors rather than individual weights. Because GReg [49] presented the results of applying the sparsity training method to both weight and channel pruning, we too apply the GSM [45] for the

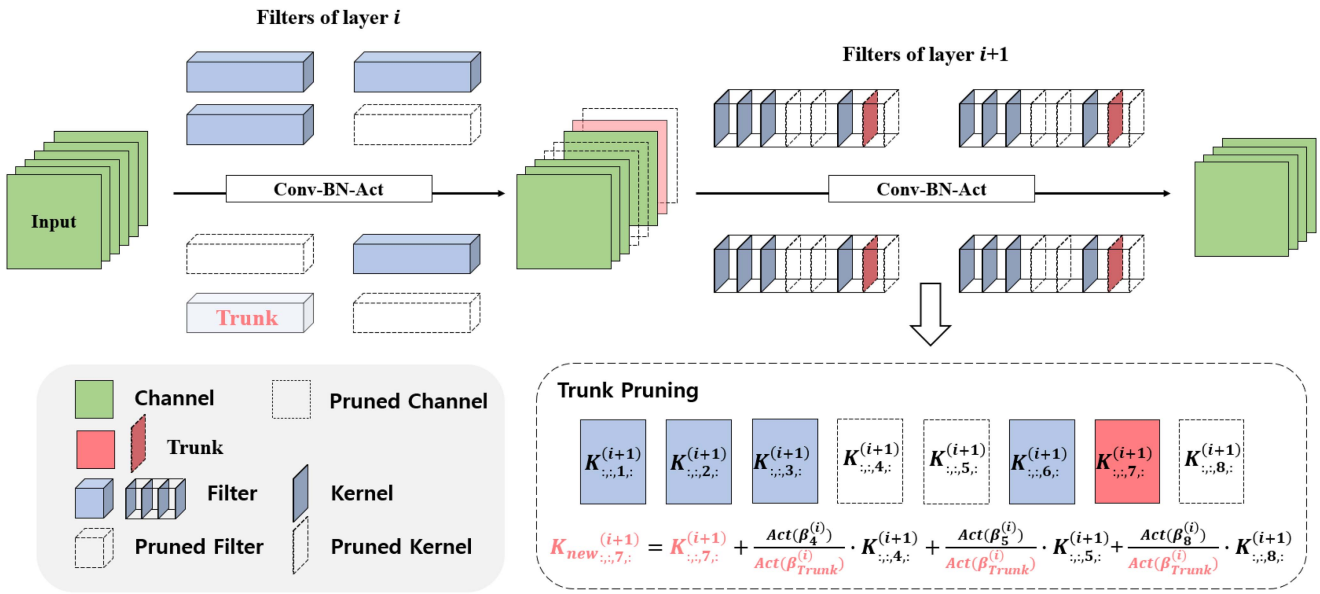
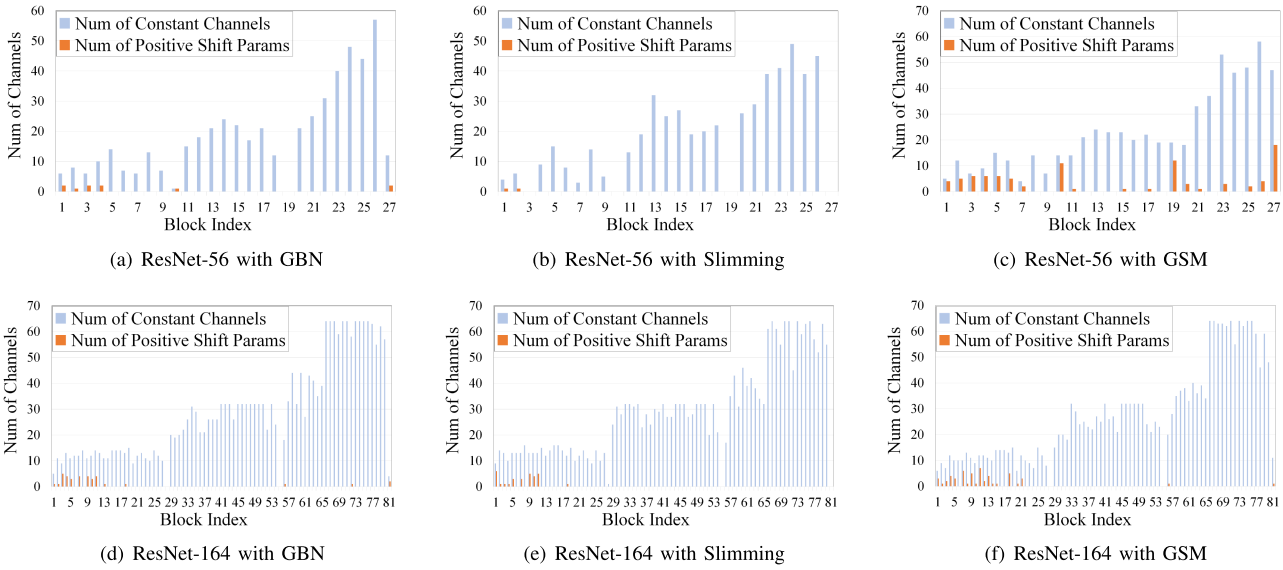


Fig. 1. Trunk pruning in two consecutive convolutional layers. The Conv-BN-Act indicates each Convolution-Batch Normalization-Activation operation. After sparsity training, the four filters (i.e., the fourth, fifth, seventh, and eighth) of layer i become redundant. When applying trunk pruning, we set the seventh filter of layer i to the trunk. Accordingly, the seventh kernel of the corresponding layer $i+1$ is also set as the trunk. Only the seventh filter of layer i remains (i.e., the fourth, fifth, and eighth filters are removed), and the trunk pruning equation is applied to the seventh kernel of layer $i+1$.

channel/filter pruning rather than the weight pruning in a similar manner. As an experimental network, we use ResNet [1], which is widely used in various applications and is difficult to prune owing to its efficient network structure. It should be noted that we only prune the internal CONV layer in the residual blocks, similar to other channel pruning methods [23], [24], [25]. However, if the input channels of the next residual block are the constant channels, trunk pruning can also be conveniently applied to the first CONV layer of the next residual block. To control the floating point operations per second (FLOPs), we use the penalty term λ suitable for the sparsity training scheme (Slimming: 5×10^{-4}, GSM: 5×10^{-5}, and GBN: 8×10^{-4}).

Table I shows the results obtained by applying the conventional and trunk pruning methods after each sparsity training using ResNet-56 and 164 on the CIFAR-10 dataset. "Baseline", "Pruned Acc. before fine-tuning", and "Pruned Acc. after fine-tuning" represent the Top-1 accuracy of the baseline, pruned networks without fine-tuning, and pruned networks with fine-tuning, respectively. "Acc. Drop before fine-tuning" and "FLOPs reduction" denote the accuracy drop between the baseline and pruned network without fine-tuning and the FLOPs reduction of the pruned network compared with the baseline, respectively. The experimental results show that trunk pruning before fine-tuning achieves the best performance in all three sparsity training methods (i.e., Slimming, GSM, and GBN). That is, because trunk pruning already has an optimally reduced representational capacity, fine-tuning has a negative effect on network performance, whereas performance recovery through fine-tuning has a certain effect when the performance degradation is large (such as in conventional pruning). It should be noted that, as mentioned in Section III-C, this negligible difference in FLOPs reduction occurs because the proposed method uses additional output channels (i.e., trunk). These results demonstrate that the proposed method is sufficiently compatible with various sparsity training methods and, thereby, provides a remarkable performance without fine-tuning. In the following subsection, we present the experimental results using the GBN as a base sparsity training method because the GBN has the best trade-off between accuracy drop and FLOPs reduction.

To analyze the cause of the accuracy drop in conventional pruning, Fig. 2 shows the distribution of the constant channels with the BN scaling parameters below the threshold (10^{-3} in our environment) for various pruning methods (GBN [34], Slimming [24], and GSM [45]). The internal layer index of the residual blocks and the number of the constant channels in ResNet-56 and ResNet-164 are displayed on the x-axis and y-axis, respectively. In ResNet-56 with GBN and Slimming, the number of positive shift parameters (orange bars) among the channels with a scaling factor below the threshold is highly marginal. The negative shift parameters (i.e., blue bar–orange bars) become zero because ResNet in Fig. 2 uses the ReLU activation function. Therefore, no performance loss occurs even when these are removed. As a result, the difference in accuracy drop between the conventional and trunk pruning methods is negligible, as shown in Table I. Meanwhile, the GSM has significantly more constant positive channels than the GBN or Slimming. Therefore, conventional pruning results in a significant accuracy drop (i.e., 4.59% vs. 0.53%). There are a number of constant positive channels (orange bars) in ResNet-164. In addition, because the network is significantly deeper compared with ResNet-56, the pruning loss in the early layer accumulates gradually in the prediction layer. This results in a significant loss. Thus, in ResNet-164, trunk pruning can achieve a significantly higher performance



(a) ResNet-56 with GBN (b) ResNet-56 with Slimming (c) ResNet-56 with GSM

(d) ResNet-164 with GBN (e) ResNet-164 with Slimming (f) ResNet-164 with GSM

Fig. 2. Distribution of the constant channels with the BN scaling parameters below the threshold (10^{-3} in our experiment) for ResNet-56 and ResNet-164 according to various sparsity training schemes (i.e., GBN, Slimming, and GSM). The blue and orange bars represent the number of constant channels (both positive and negative) and the number of constant channels with only positive values, respectively. The x-axis and y-axis represent the internal layer index and number of constant channels of the residual block, respectively. FLOPs reduction and accuracy corresponding to the number of channels of each sub-figure are presented in Table I Among the three sparsity training techniques, the Slimming method with the lowest positive shift parameter ratio causes the greatest decrease in accuracy. Additionally, a greater decrease in accuracy occurs in ResNet-56, where the proportion of positive shift parameters is lower than in ResNet-164. (a) ResNet-56 with GBN (b) ResNet-56 with Slimming (c) ResNet-56 with GSM (d) ResNet-164 with GBN (e) ResNet-164 with Slimming (f) ResNet-164 with GSM.

than conventional pruning with a marginal FLOPs loss. This is because it can reproduce the output of the unpruned network entirely, so that the pruning loss is not transferred to the prediction layer.

C. Comparison Results for Various Networks With the CIFAR-10 and CIFAR-100 Datasets

Table II shows the results of the performance comparison using ResNet-56, ResNet-164, and VGG-19 [38], which are the representative networks for image classification, on the CIFAR-10 dataset. As mentioned in the previous subsection, "Ours" denotes the results obtained by applying the trunk pruning without fine-tuning after the GBN-based sparsity training. "Fine-Tuning?" indicates whether the pruned network performed fine-tuning or not. The "-" in the results of NISP [50] denotes "Not provided in the reference article." In all the comparison result tables, we present the baseline accuracy results of the comparative studies without modification because the network baseline accuracy presented in each study varies owing to different environments (e.g., experimental setting and implementation language). Accordingly, to perform a fair comparison, we compare the performance in terms of the trade-off between relative accuracy drop (Drop (%)) and FLOPs reduction (Pruned FLOPs (%)), rather than absolute accuracy (i.e., Pruned Acc. (%)).

The proposed pruned ResNet-56 with 50% FLOPs reduction enhances the Top-1 accuracy by 0.01% over the baseline and shows better performance than all the existing studies, including the SOTA channel pruning methods that require time-consuming fine-tuning. Moreover, the pruned ResNet-56 with trunk pruning shows a higher FLOPs reduction as well as a lower accuracy drop compared with Variational [29], without fine-tuning. As mentioned in Section II-C, Variational [29] reformulates the BN to remove additional fine-tuning. However, because this reformulation is not an affine function of the existing BN, the flow and value of gradients are changed significantly and cannot ensure optimal performance. Meanwhile, the proposed method can achieve a better performance because it can remove fine-tuning without modifying the affine function of the existing BN. For ResNet-164, we reduce 6.51% more FLOPs with a 0.02% lower accuracy drop than C-SGD [27]. This is because the proposed method can automatically identify the optimal structure (i.e., optimal layer width) by using sparsity training. However, C-SGD also removes the channels of important layers by using uniform pruning (i.e., removing the same portion for all the layers), which causes performance degradation. In other words, the two methods may be similar in that fine-tuning is unnecessary, but the basic rationale is different in terms of approaches to solving this problem. It should also be noted that the pruned ResNet-164 of C-SGD [27] requires three times more training (1,800 epochs (total of 600-600-600) with a batch size of 64). However, we use only one-shot pruning. It is significantly more efficient in terms of training time. The proposed pruned VGG-19 shows better accuracy in reducing the FLOPs similar to SCP [31]. Furthermore, it achieves an accuracy improvement of 0.16% compared with the baseline, although it prunes over 70% of the FLOPs. It should be noted that when pruning an overparameterized model using a high-performance sparsity training method, the generalization of the model can be improved. Consequently, we can

generate a sparse model with high accuracy. Since modern neural networks are highly overparameterized, these networks are highly vulnerable to overfitting to the training data. Pruning can solve the problem of overparameterization by reducing the complexity of the model by removing unnecessary parameters. As a result, a pruned model can become less sensitive to the noise in the training data and learn more general patterns, leading to improved generalization performance [43], [51]. To conclude, performing the proposed trunk pruning after the sparsity training enables a high FLOPSs reduction and high accuracy even when the existing fine-tuning process is not performed.

Table III shows the results of the performance comparison using ResNet-56 on the CIFAR-100 dataset. ResNet-56 with trunk pruning also shows the smallest accuracy drop with similar FLOPs reduction compared to other comparison studies. The results of these CIFAR-100 datasets demonstrate that the proposed method is an efficient pruning method on various datasets.

D. Comparison Results on Various Networks With the ILSVRC-2012 Dataset

We demonstrate the superiority of the proposed trunk pruning on the ImageNet ILSVRC-2012 dataset using ResNet-50, ResNet101, and MobileNet-V1 [2]. These are used frequently in object detection and segmentation. As shown in Table IV, the trunk pruning with the GBN [34] also achieves SOTA performance on ResNet-50 and ResNet-101 in terms of FLOPs reduction and accuracy drop, even in the challenging ImageNet ILSVRC-2012 dataset. With ResNet-50, the proposed method improves the accuracy over the baseline by 0.34%, although it achieves a FLOPs reduction (i.e., 39.29%) similar to that of Taylor-FO [28] and ResRep [56]. For Taylor-FO, the channel importance evaluation method is similar to trunk pruning. However, it performs an iterative process of pruning and fine-tuning. It should be noted that this process is time-consuming and straightforwardly falls into a bad local minima owing to fine-tuning [27], [31], [36], which causes significant performance degradation. Furthermore, the proposed method with approximately 60% FLOPs reduction mitigates the accuracy drop by 0.37% compared to ResRep [56]. It is noteworthy that the proposed method has better pruning performance than all existing methods that require fine-tuning. In addition, the pruned ResNet-101 is superior to Rethinking [26] and Taylor-FO [28] in terms of the trade-off between accuracy and FLOPs reduction. It should be noted that the two "Ours" in ResNet-50 are the results of trunk pruning using different sparsity strengths under an identical threshold. Consequently, each "Ours" displays a different FLOPs reduction, although the threshold is identical.

Next, Table V shows the results of the pruned MobileNet-V1. The "-" in the results of NetAdapt [62] denotes "Not provided in the reference article". MobileNet-V1 [2] is known to be generally difficult to be pruned because it uses depthwise convolution, unlike existing networks. The experimental results show that the proposed trunk pruning achieves a reasonable level of accuracy drop without fine-tuning and better performance than the existing method [62] with fine-tuning. Meanwhile, MobileNet-V1 pruned by conventional pruning does not preserve accuracy, and additional fine-tuning is essential.

E. Verification of Compatibility With Various Activation Functions Including Negative Values

a) Trunk vs. Conventional: In this subsection, we demonstrate the effectiveness of trunk pruning using the activation functions with negative values (Mish [41], Swish [42], and LeakyReLU). The effect of a change in the activation function on the proposed method is a change in the shift parameters included in the feature maps generated from the filter to be removed, and the proposed method is a method of preserving these shift parameters. Therefore, it should be noted that in the process of implementing new activation functions such as Mish, Swish, and Leaky activation functions, we can simply replace the existing ReLU with these new activation functions without any additional process. Because the number of negative constant channels dominates among all the constant channels, as shown in Fig. 2, if the activation functions with negative values (such as Mish, Swish, and Leaky) are used, the loss of all the constant channels is transferred to the next layer in the conventional pruning methods. This causes a severe accuracy drop in the pruned network without fine-tuning. However, as shown in Table VI, the trunk pruning almost perfectly complements the performance degradation caused by the activation functions with negative values. It is noteworthy that this tendency appears much more clearly in ResNet-50 because ResNet-50 is a much larger model than ResNet-56.

b) Comparison with Other Pruning Methods: We compare the performance by applying the Mish and Swish activation functions to the C-SGD [27] and AOFP [63], which are not limited to ReLU. Table VII shows the results for the pruned ResNet-56. "Training Epochs" refers to the number of training epochs. For example, 300 + 150 + 150 in AOFP corresponds to baseline + training for pruning + fine-tuning epochs. We also measure the training time for each method. All training processes are performed fairly using one Geforce RTX 2080, and all experimental results are measured including all training times of baseline, pruning, and fine-tuning. The experimental results demonstrate that the proposed method with Mish activation achieves better performance in terms of the trade-off between FLOPs reduction and accuracy drop compared with the other two methods. It also shows an advantage in terms of training time. In particular, in the results of the Mish activation function, trunk pruning mitigates an accuracy drop of 0.22% compared with C-SGD while reducing 1.5 hours of training time. Even for the swish activation function, trunk pruning achieves a significantly large accuracy improvement (i.e., 1.01%) compared with C-SGD for an equal training time (i.e., 6 hours). These results show that the proposed method has better compatibility with various activation functions due to the basic rationale difference between the proposed method and C-SGD. It is noteworthy that other channel pruning studies also evaluated the efficiency of the proposed method based on the results for training time in the GPU rather than training epoch [64]. Moreover, the proposed method using Swish activation shows a performance similar to that of C-SGD

TABLE VIII
TRANSFER LEARNING RESULTS WITH THE YOLOV3 ON PASCAL VOC 2007

Network	Input Size	Backbone FLOPs Reduction (%)	mAP@0.5 (%)	FPS
Unpruned (Backbone: Original DrakNet-53	416	-	79.68	104
Unpruned (Backbone: Original DrakNet-53	416	**44.7**	**79.16**	**134**
Unpruned (Backbone: Original DrakNet-53	416	42.9	78.9	134

The bold values represent the key results of our experiments.

TABLE IX
COMPARISON OF ACCURACY AND FLOPS REDUCTION ACCORDING TO VARIOUS THRESHOLDS

Network	Threshold	Top-1 Accuracy (%)	FLOPs reduction (%)
Res56	{0.001, 0.01, 0.1}	{93.59, 93.59, 42.3}	{50.4, 50.6, 51.2}
Res164	{0.001, 0.01, 0.1}	{94.22, 94.22, 94.16}	{67.4, 67.4, 67.5}
VGG19	{0.001, 0.01, 0.1}	{93.14, 91.56, 10.0}	{73.4, 73.4, 86.8}
Res50(ReLU)	{0.001, 0.01, 0.1}	{75.62, 75.62, 0.11}	{60.9, 60.9, 86.1}
Res50(Mish)	{0.001, 0.01, 0.1}	{76.41, 76.41, 75.88}	{50.9, 50.9, 51.0}
Res50(Swish)	{0.001, 0.01, 0.1}	{76.70, 76.70, 67.41}	{51.8, 51.8, 53.7}

TABLE X
ABLATION STUDY RELATED TO THE NUMBER OF FILTERS

Network	# of Total filters	# of Removed filters	# of Trunk filters	# of Trunk / # of Total filters	# of Trunk / # of Removed filters
Res56	2128	497	6	0.28%	1.21%
Res164	6160	2037	14	0.23%	0.69%
VGG19	5504	4651	9	0.16%	0.19%
Res50(ReLU)	26560	4522	32	0.12%	0.71%
Res50(Mish)	26560	3741	33	0.12%	0.88%
Res50(Swish)	26560	3718	33	0.12%	0.88%

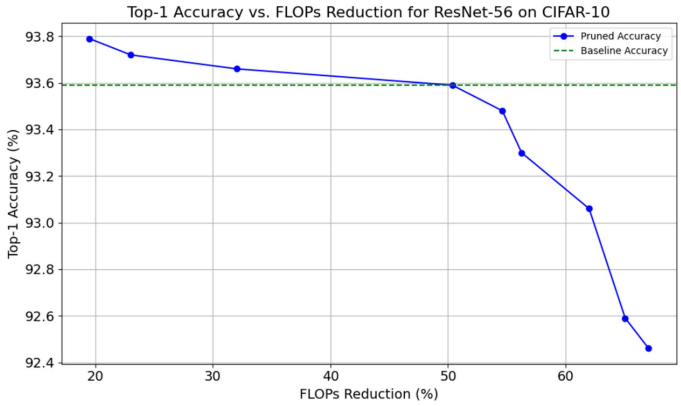


Fig. 3. Top-1 accuracy (%) for various FLOPs reduction using ResNet-56 on the CIFAR-10 dataset.

in the same training epochs and is more efficient in terms of training time.

F. Scalability of the Proposed Method to the Object Detection Task

We also test the scalability of the proposed trunk pruning on the object detection task by using transfer learning. The experiments are conducted using the YOLOv3 [8] (Backbone: DarkNet-53) on the PASCAL VOC dataset with 20 classes. It is a representative one-stage object detection model. We prune all the layers except the first layer of DarkNet-53 on the ImageNet ILSVRC-12 dataset [40]. As shown in Table VIII, we present the mAP@0.5 and frames per second (FPS) of YOLOv3 before and after backbone pruning. The pruned YOLOv3 using the proposed method achieves a significant FPS improvement (28.8%, measured by the Geforce RTX 2080 Ti GPU-server) with a small mAP drop.

G. Ablation Studies

Table IX presents a performance comparison experiment according to various thresholds using ResNet-56, ResNet-164, and VGG-19 on CIFAR-10 and ResNet-50 with three activation functions on ImageNet as an ablation study. Specifically, we present the Top-1 accuracy and FLOPs reduction according to various thresholds. In general, a high FLOPs reduction is achieved at a high threshold, although a large accuracy drop may occur accordingly. However, the effect of FLOPs reduction (which increases as the threshold increases) is relatively smaller than the accuracy drop. That is, although the difference in FLOPs reduction is not noteworthy, a threshold of 0.1 has a significant negative impact on the performance, whereas thresholds of 0.001 and 0.01 have negligible effects on the accuracy. However, in the case of VGG19, an additional 1.58% drop occurs with the same FLOPs reduction at 0.01 compared to 0.001. Therefore, we empirically use the threshold of 0.001 based on these results to focus on the Top-1 accuracy drop. This also implies that it is reasonable to approximate a sufficiently small scaling factor to zero.

In addition, we present the number of filters in Table X to verify the influence of the trunk filter. In ResNet-56, ResNet-164, and VGG-19 on CIFAR-10, the proposed method removes 497, 2037, and 4651 filters out of a total of 2128, 6160, and 5504 filters, respectively. It additionally uses 6 (=0.28% of the total number of filters, 1.21% of the number of removed filters), 14 (=0.23% of the total number of filters, 0.69% of the number of removed filters), and 9 (=0.16% of the total number of filters, 0.19% of the number of removed filters) trunks, respectively. It can be observed that the number of trunk filters is significantly smaller compared with the total number of filters and number of removed filters. Consequently, the negative effect of the FLOPs reduction owing to the additional trunk filters is negligible. In addition, ResNet-50 on ImageNet has a smaller ratio of additional trunk filters to total filters than ResNet-56/164 regardless of the activation function. This means that the negative effect of FLOPs reduction due to the additional trunk filter is more negligible in ResNet-50.

Lastly, to show the accuracy change of trunk pruning according to different pruning rates, Fig. 3 shows top-1 accuracy according to various pruning rates (i.e., FLOPs reduction) of about 19% to 67% on the CIFAR-10 dataset using ResNet-56 as an ablation study. The top-1 accuracy of pruned ResNet-56 decreases linearly as the pruning rate increases. Additionally, even when the pruning rate increases, especially at a high pruning rate of 67%, the accuracy of 92.46% is maintained, showing relatively little performance degradation. Additionally, at pruning rates between 19% and 50%, the accuracy is higher than the baseline accuracy. This shows that the proposed trunk pruning has a network regularization effect.

V. Conclusion

This article proposes trunk pruning to eliminate the time-consuming fine-tuning process in channel pruning based on sparsity training. The proposed trunk pruning incorporates the information of the kernels to be pruned in the next layer into one kernel (the trunk), considering the effect of the batch normalization shift parameter remaining after the sparsity training. Thereby, the pruned network can reproduce the output of the unpruned network after the sparsity training. This results in the elimination of the fine-tuning process. In addition, trunk pruning is highly compatible with various sparsity training methods and can also be applied to activation functions with negative values. It is feasible to achieve optimal performance by combining this method with sparsity training and several activation functions.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*., 2016, pp. 770–778.

[2] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*., 2017, pp. 4700–4708.

[4] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[5] C.-Y. Wang et al., "CSPNet: A new backbone that can enhance learning capability of CNN," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2020, pp. 390–391.

[6] A. Howard et al., "Searching for MobileNetV3," in *Proc. IEEE Int. Conf. Comput. Vis*., 2019, pp. 1314–1324.

[7] J. Choi, D. Chun, H. Kim, and H.-J. Lee, "Gaussian YOLOv3: An accurate and fast object detector using localization uncertainty for autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis*., 2019, pp. 502–511.

[8] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.

[9] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis*., 2015, pp. 1440–1448.

[10] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-YOLOv4: Scaling cross stage partial network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2021, pp. 13029–13038.

[11] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10781–10790.

[12] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT: Real-time instance segmentation," in *Proc. IEEE Int. Conf. Comput. Vis*., 2019, pp. 9157–9166.

[13] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis*., 2017, pp. 2961–2969.

[14] J. Choi, D. Chun, H.-J. Lee, and H. Kim, "Uncertainty-based object detector for autonomous driving embedded platforms," in *Proc. IEEE 2nd Int. Conf. Artif. Intell. Circuits Syst.*, 2020, pp. 16–20.

[15] D. T. Nguyen, H. Kim, H.-J. Lee, and I.-J. Chang, "An approximate memory architecture for a reduction of refresh power consumption in deep learning applications," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2018, pp. 1–5.

[16] D. T. Nguyen, T. N. Nguyen, H. Kim, and H.-J. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 27, no. 8, pp. 1861–1873, Aug. 2019.

[17] J. Wang et al., "SmsNet: A new deep convolutional neural network model for adversarial example detection," *IEEE Trans. Multimedia*, vol. 24, pp. 230–244, 2022.

[18] S. Kim and H. Kim, "Zero-centered fixed-point quantization with iterative retraining for deep convolutional neural network-based object detectors," *IEEE Access*, vol. 9, pp. 20828–20839, 2021.

[19] D. T. Nguyen, H. Kim, and H.-J. Lee, "Layer-specific optimization for mixed data flow with mixed precision in FPGA design for CNN-based object detectors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 6, pp. 2450–2464, Jun. 2021.

[20] H.-J. Kang, "Accelerator-aware pruning for convolutional neural networks," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 7, pp. 2093–2103, Jul. 2020.

[21] Z. Wang, W. Hong, Y.-P. Tan, and J. Yuan, "Pruning 3D filters for accelerating 3D convNets," *IEEE Trans. Multimedia*, vol. 22, pp. 2126–2137, 2020.

[22] Y. Xu, W. Dai, Y. Qi, J. Zou, and H. Xiong, "Iterative deep neural network quantization with Lipschitz constraint," *IEEE Trans. Multimedia*, vol. 22, pp. 1874–1888, 2020.

[23] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convNets," in *Proc. 5th Int. Conf. Learn. Representations*, 2017.

[24] Z. Liu et al., "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis*., 2017, pp. 2736–2744.

[25] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*., 2019, pp. 4340–4349.

[26] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," in *Proc. 6th Int. Conf. Learn. Representations*, 2018.

[27] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal SGD for pruning very deep convolutional networks with complicated structure," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*., 2019, pp. 4943–4953.

[28] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*., 2019, pp. 11264–11272.

[29] C. Zhao et al., "Variational convolutional neural network pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*., 2019, pp. 2780–2789.

[30] Y. He et al., "Learning filter pruning criteria for deep convolutional neural networks acceleration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*., 2020, pp. 2009–2018.

[31] M. Kang and B. Han, "Operation-aware soft channel pruning using differentiable masks," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5122–5131.

[32] J. Guo, W. Zhang, W. Ouyang, and D. Xu, "Model compression using progressive channel pruning," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 3, pp. 1114–1124, Mar. 2021.

[33] N. J. Kim and H. Kim, "FP-AGL: Filter pruning with adaptive gradient learning for accelerating deep convolutional neural networks," *IEEE Trans. Multimedia*, vol. 25, pp. 5279–5290, 2023.

[34] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 2133–2144.

[35] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[36] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *Proc. Int. Conf. Learn. Representations*, 2019.

[37] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *Proc. 8th Int. Conf. Learn. Representations*, 2020.

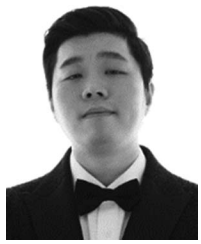
[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[39] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.

[40] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[41] D. Misra, "Mish: A self regularized non-monotonic neural activation function," in *Proc. 31st Brit. Mach. Vis. Conf.*, 2020.
[42] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017, *arXiv:1710.05941*.
[43] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. 2nd Int. Conf. Neural Inf. Process. Syst.*, 1990, pp. 598–605.
[44] S. Han et al., "DSD: Dense-sparse-dense training for deep neural networks," in *Proc. 5th Int. Conf. Learn. Representations*, 2017.
[45] X. Ding et al., "Global sparse momentum SGD for pruning very deep neural networks," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 6382–6394.
[46] T. Zhuang et al., "Neuron-level structured pruning using polarization regularizer," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 9865–9877.
[47] Z. Liu et al., "Metapruning: Meta learning for automatic neural network channel pruning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 3296–3305.
[48] M. Lin et al., "Channel pruning via automatic structure search," in *Proc. 29th Int. Conf. Int. Joint Conf. Artif. Intell.*, 2021, pp. 673–679.
[49] H. Wang, C. Qin, Y. Zhang, and Y. Fu, "Neural pruning via growing regularization," in *Proc. Int. Conf. Learn. Representations*, 2021.
[50] R. Yu et al., "NISP: Pruning networks using neuron importance score propagation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 9194–9203.
[51] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. 7th Int. Conf. Learn. Representations*, 2019.
[52] X. Ning et al., "DSA: More efficient budgeted pruning via differentiable sparsity allocation," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 592–607.
[53] H. Peng, J. Wu, S. Chen, and J. Huang, "Collaborative channel pruning for deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5113–5122.
[54] M. Lin et al., "HRank: Filter pruning using high-rank feature map," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1529–1538.
[55] Y. He, P. Liu, L. Zhu, and Y. Yang, "Filter pruning by switching to neighboring CNNs with good attributes," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 10, pp. 8044–8056, Oct. 2023.
[56] X. Ding et al., "ResRep: Lossless CNN pruning via decoupling remembering and forgetting," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 4510–4520.
[57] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 2234–2240.
[58] Z. Wang et al., "QSFM: Model pruning based on quantified similarity between feature maps for AI on edge," *IEEE Internet Things J.*, vol. 9, no. 23, pp. 24506–24515, Dec. 2022.
[59] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 304–320.
[60] M. Lin et al., "Network pruning using adaptive exemplar filters," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 12, pp. 7357–7366, Dec. 2022.
[61] J. Liu et al., "Discrimination-aware network pruning for deep model compression," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 8, pp. 4035–4051, Aug. 2022.
[62] T.-J. Yang et al., "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 285–300.
[63] X. Ding, G. Ding, Y. Guo, J. Han, and C. Yan, "Approximated oracle filter pruning for destructive CNN width optimization," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1607–1616. [Online]. Available: https://github.com/DingXiaoH/AOFP
[64] B. Li, B. Wu, J. Su, and G. Wang, "Eagleeye: Fast sub-net evaluation for efficient neural network pruning," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 639–654.

Nam Joon Kim (Graduate Student Member, IEEE) received the B.S. degree in electrical and information engineering from the Seoul National University of Science and Technology, Seoul, Korea, in 2019. He is currently an M.S. Student in electrical and information engineering with the Seoul National University of Science and Technology. His research focuses on the areas of network pruning, quantization, and efficient network design for deep neural networks.



Hyun Kim (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, Korea, in 2009, 2011 and 2015, respectively. From 2015 to 2018, he was with the BK21 Creative Research Engineer Development for IT, Seoul National University, as a BK Assistant Professor. In 2018, he was with the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, where he is currently an Associate Professor. His research interests include the areas of algorithm, computer architecture, memory, and SoC design for low-complexity multimedia applications and deep neural networks.