Received 19 September 2023, accepted 18 October 2023, date of publication 23 October 2023, date of current version 31 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3326534

RESEARCH ARTICLE

V-SKP: Vectorized Kernel-Based Structured Kernel Pruning for Accelerating Deep Convolutional Neural Networks

KWANGHYUN KOO, (Graduate Student Member, IEEE), AND HYUN KIM, (Senior Member, IEEE)
Department of Electrical and Information Engineering, Research Center for Electrical and Information Technology, Seoul National University of Science and Technology, Seoul 01811, South Korea

Corresponding author: Hyun Kim (hyunkim@seoultech.ac.kr)

This work was supported by the Ministry of Science and ICT (MSIT), South Korea, through the Information Technology Research Center (ITRC) Support Program, supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP), under Grant IITP-2023-RS-2022-00156295.

ABSTRACT In recent years, kernel pruning, which offers the advantages of both weight and filter pruning methods, has been actively conducted. Although kernel pruning must be implemented as structured pruning to obtain the actual network acceleration effect on GPUs, most existing methods are limited in that they have been implemented as unstructured pruning. To compensate for this problem, we propose vectorized kernel-based structured kernel pruning (V-SKP), which has a high FLOPs reduction effect with minimal reduction in accuracy while maintaining a 4D weight structure. V-SKP treats the kernel of the convolution layer as a vector and performs pruning by extracting the feature from each filter vector of the convolution layer. Conventional L1/L2 norm-based pruning considers only the size of the vector and removes parameters without considering the direction of the vector, whereas V-SKP removes the kernel by considering both the feature of the filter vector and the size and direction of the vectorized kernel. Moreover, because the kernel-pruned weight cannot be utilized when using the typical convolution, in this study, the kernel-pruned weights and the input channels are matched by compressing and storing the retained kernel index in the kernel index set during the proposed kernel pruning scheme. In addition, a kernel index convolution method is proposed to perform convolution operations by matching the input channels with the kernel-pruned weights on the GPU structure. Experimental results show that V-SKP achieves a significant level of parameter and FLOPs reduction with acceptable accuracy degradation in various networks, including ResNet-50, and facilitates real acceleration effects on the GPUs, unlike conventional kernel pruning techniques.

INDEX TERMS Kernel pruning, convolutional neural networks, vectorized kernel, network compression.

I. INTRODUCTION

Deep convolutional neural networks (DCNNs) have afforded significant achievements in computer vision fields, such as image classification [1], object detection [2], [3], and semantic segmentation [4], [5], and most related studies [6] have accumulated deep hidden layers gradually to improve the performance of DCNNs. However, as layer depth increases, the number of parameters in the network increases, and the required computational costs and memory capacity/accesses increase rapidly, rendering DCNNs impractical. Hence, the size of heavy weights must be reduced to enable the practical use of DCNNs in power-restricted environments, such as mobile devices [7], [8].

In recent years, most networks are over-parameterized, and as the size of the network increases, the proportion of unnecessary parameters increases [9]. The network pruning method [10], [11], which removes these unnecessary parameters, yields the best performance compared with other network compression methods in terms of the trade-off

The associate editor coordinating the review of this manuscript and approving it for publication was Mingbo Zhao.

between network compression effect and accuracy. Two main methods of pruning exist: unstructured [12], [13], [14], [15], [16] and structured [17], [18], [19], [20], [21], [22], [23], [24], [25]. The criteria for classifying unstructured and structured pruning are generally determined based on the pruning unit in the network. Weight pruning, which is the most representative unstructured pruning method, compresses the network by assigning a value of zero to the weights to be removed based on the pruning criterion. Because weight pruning is performed in the smallest unit of the networks (*i.e.*, weight unit), it can be applied flexibly and is not constrained by the network structure. However, it is difficult to accelerate DCNNs in the graphics processing unit (GPU) environment via weight pruning, because the weights assigned to zero value are not removed structurally. By contrast, filter pruning, a representative method of structured pruning, selects a filter to be removed based on a pruning criterion and structurally removes the filter from the network [26]. Unlike weight pruning, filter pruning can be easily accelerated in a GPU environment [27], and its acceleration performance is high because the actual operation is reduced through structural removal. However, because the filter corresponds to a unit larger than the weight, its application in complicated network structures is limited. For example, in the case of a network with a shortcut layer, such as ResNet [1], the number of output channels must be the same because the two layers connected to the shortcut layer are added to each other's outputs. However, if filter pruning is applied, the number of output channels of the layer is reduced, and consequently, the number of output channels of the two layers connected by the shortcut layer changes, resulting in an open problem [22] that hinders summation operations.

Kernel pruning, an intermediate step between weight pruning and filter pruning, is applied based on the kernel, which is the smallest structural unit in the network. Kernel pruning can be classified into the following categories. Structured kernel pruning is a technique that systematically removes targeted kernels, while unstructured kernel pruning involves replacing the targeted kernels with zero values. Additionally, the partially structured kernel pruning is a hybrid approach that combines the characteristics of both unstructured and structured kernel pruning methods. In general, this approach has the advantage of enabling real acceleration by reducing unnecessary kernels. Unlike filter pruning, kernel pruning can be free from open problems, because the number of output channels is maintained, which means that it can be flexibly applied to networks. Kernel pruning can be applied unproblematically to the shortcut layer of ResNet [1] and MobileNet-V2 [28], which typically have open problems. However, if the kernel is simply removed through the pruning criterion, a different ratio of kernels is removed for each filter, and the layer's 4D weight structure collapses. In addition, even if the kernel is removed while maintaining the 4D weight structure of the layer, the convolution operation cannot be performed, because it cannot correspond one-to-one with the kernel in the filter and the input feature map of the filter. Therefore, it is not easy to implement structured kernel pruning, and since it causes relatively serious performance degradation, it is implemented as unstructured kernel pruning in most cases [29]. However, because no structural change occurs in the layer weight even after unstructured kernel pruning is performed, the actual acceleration effect cannot be achieved on the GPU as in weight pruning.

To overcome the limitations of the existing kernel pruning, this study proposes a vectorized kernel-based structured kernel pruning (V-SKP) method. In V-SKP, by treating the kernel of the convolution layer as a vector, the importance score of each kernel is obtained through the L1 norm of the kernel and the feature of the filter vector, which is the sum of all vectorized kernels of one filter. To reflect the different sensitivities of each layer, we use this importance score to implement global threshold-based global pruning [30]. While conventional L1/L2 norm-based pruning considers only the size of the vectorized kernel, V-SKP removes the kernel by considering the direction of the filter vector as well as the size of the vectorized kernel. In addition, V-SKP is implemented with structure-aware kernel pruning by removing kernels in the same ratio from the filter, which is a three-dimensional value, to maintain the layer weight structure of 4-dimensional (4D) arrays. If structured kernel pruning is performed while maintaining the layer weights of a 4D array, a kernel-pruned weight is obtained. Each filter in the kernel-pruned weight must receive a corresponding input feature map, but since the number of kernels in the filter of the kernel-pruned weight has been reduced, it cannot correspond one-to-one with the input feature map. In the end, the convolution operation can be performed in correspondence with the input feature map only when the indexes of the remaining kernels are known. To solve this problem, in this paper, kernel index convolution using kernel index set is also proposed, and the acceleration effect is verified by realizing this method through CUDA. The superior performance of the proposed V-SKP compared to the existing pruning studies is verified in various datasets and models, and it is confirmed that the acceleration effect of 2.72× can be achieved through the actual latency test. The contributions of this study can be summarized as follows:

- To achieve a high pruning rate without degrading network performance, a kernel pruning technique that considers both the size and direction of each kernel in the filter is proposed through the vectorized kernel.
- Structured kernel pruning is implemented by adaptively reflecting the importance of each layer while maintaining the 4D weight structure of the weight parameter.
- Kernel-pruned weights, which cannot be calculated with general convolution, can be calculated through kernel index convolution.

The remainder of this paper is organized as follows. Section II introduces the concept of kernel pruning and related existing studies. Section III presents a detailed description of the proposed V-SKP, and Section IV provides the experimental results to evaluate the proposed method. Finally, Section V concludes this paper.

II. RELATED WORKS: KERNEL PRUNING

In a representative study pertaining to unstructured kernel pruning [29], a single-port salient mapping channel and a dual-port salient mapping channel were obtained via pruning training. The single-port salient mapping channel refers to convolutional kernels with strong mapping ability in the single direction (*i.e.*, input-output direction or output-input direction), and the dual-port salient mapping channel refers to a convolutional kernel with strong mapping ability in both directions. The previous work [29] presents a kernel pruning method that defines the average mapping ability measurement index and sets the retained kernel set as either a single-port or dual-port salient mapping channel, compared with the threshold. In [31], another example of unstructured kernel pruning is presented, where the connectivity between the input channel and the kernel corresponding to the channel is evaluated based on the L2 norm. However, in these studies, structured pruning cannot be implemented because the 4D array weight structure cannot be maintained since structured pruning is not considered.

KSE [32], a partially structured kernel pruning method, measures the sparsity and entropy of the input feature map and partitions the kernel corresponding to the input feature map into important and insignificant sets via clustering. KSE implements partially structured kernel pruning by removing unimportant sets and not receiving the corresponding input feature map. While KSE maintains the 4D array weight structure by removing the kernel at the same location from all filters in a layer, it cannot effectively extract features for the entire input because it receives only a portion of the input feature map, and therefore, actual acceleration cannot be achieved in GPUs. As another study of the structured kernel pruning technique, TMI-GKP [33] uses the filter group formed by clustering filters related to each other through TMI scores for group convolution, and removes kernels of the same order from each filter in the group. Then, structured kernel pruning is implemented by removing the same percentage of kernels from other filter groups of the layer. However, removing kernels in the same order from multiple filters can cause performance degradation because the properties of each filter cannot be effectively maintained, and this method also has a limitation in that the number of input channels is increased compared to the previous one because the input channel must be newly configured according to the input channel combination suitable for the filter group due to group convolution. PPT-KP [34] addressed the problems of existing kernel pruning methods by securing multiple margin spaces for kernel pruning through an adaptive regularizer that applies different strengths of L1 regularization depending on the size of the L1 norm of the kernel. However, this method has limitations in that it requires an additional training process to secure multiple margin spaces and has not been verified on a large dataset such as ImageNet ILSVRC-2012 dataset [35].

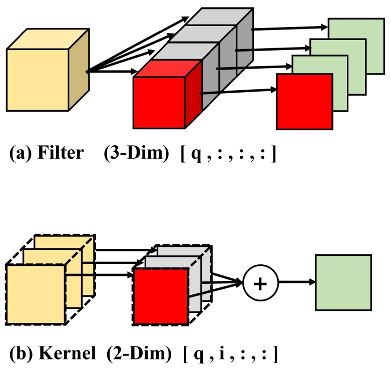

(a) Filter (3-Dim) [q , : , : , :]
(b) Kernel (2-Dim) [q , i , : , :]
+

FIGURE 1. Structure of pruning. (a) Filter pruning, (b) Kernel pruning. If the filter is removed in (a), the output channel of the filter is also removed. In contrast, even if the kernel is removed in (b), the output channel of the filter is maintained.

III. VECTORIZED KERNEL-BASED STRUCTURED KERNEL PRUNING

A. OVERALL STRUCTURE

The backbone structure of the DCNN network is generally a structure in which the layer weights of a 4D array are stacked. In the layer, the 3D component weight is a filter (as shown in Fig. 1(a)), and each filter produces an output feature map of one channel. Thus, the number of filters in the convolution layer is the same as the number of output channels in the corresponding layer. On the other hand, as shown in Fig. 1(b), the two-dimensional component weight in the layer is the kernel, and the same number of kernels as the input feature map exists in one filter. Each kernel performs a convolution operation in correspondence with one input feature map channel. As a result, the output feature map of the filter is the sum of all the results obtained by convolution operations on the kernel and the input feature map in the filter.

Kernel pruning is a method of removing such a kernel, and since doing so does not change the number of filters, the number of channels in the output feature map is maintained. The method proposed in this study is shown in Fig. 2, divided into two parts: the pruning part and the process to convolution the kernel-pruned weight. The pruning part of Fig. 2(a) is summarized as follows. The proposed vectorized kernel pruning removes the kernel while maintaining the 4D array structure of the network and recovers performance through fine-tuning. Then, the remaining kernels are structurally reconstructed and the indexes of the remaining kernels are stored in the kernel index set (see SectionsIII-B & III-C for details). The process of inference of the kernel-pruned weight in Fig. 2(b) is summarized as follows. Kernel-pruned weights have a problem that cannot be used for general convolution, and a method called kernel index convolution is proposed to solve the problem (see SectionsIII-C & III-D for details).

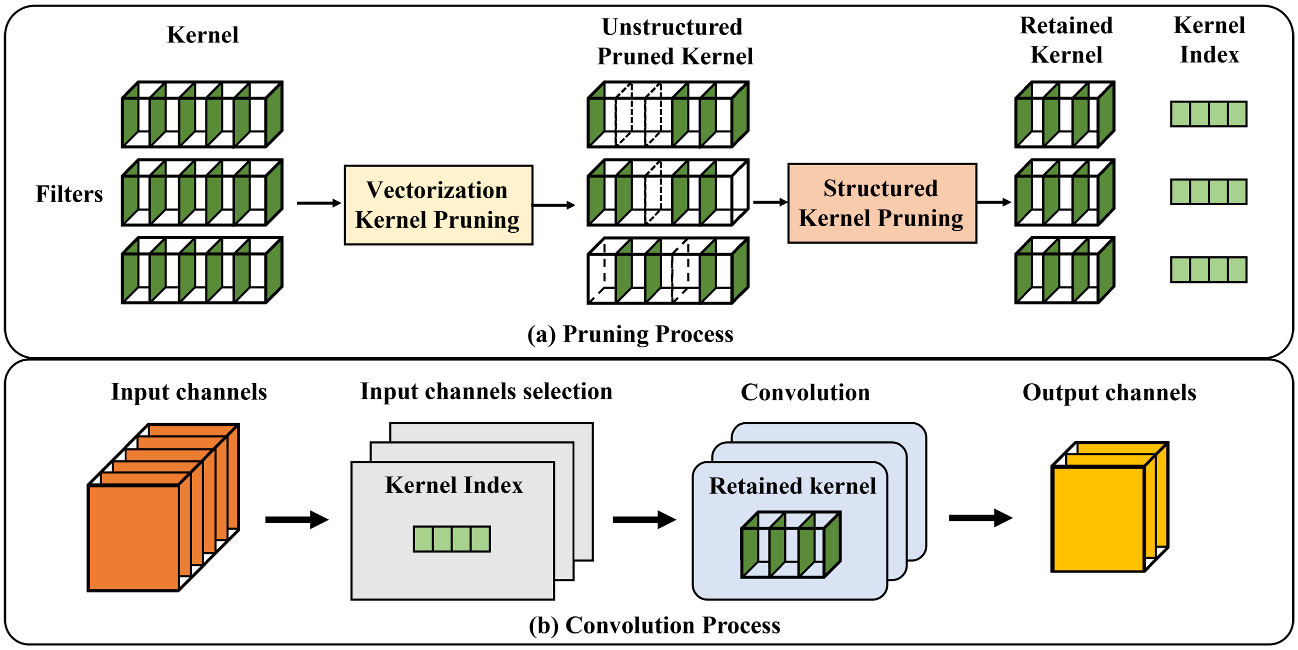


FIGURE 2. Schematic diagram of the proposed method. The proposed method is divided into (a) pruning part and (b) convolution part. (a) shows the overall process of the proposed vectorized and structured pruning, and (b) shows the overall process of convolution in the pruned network in kernel units.

B. VECTORIZED KERNEL PRUNING

In this study, vectorization is performed to make various attempts in evaluating the properties of the kernel. Through the vectorization of the kernel, it is possible to consider not only the L1 norm, which was previously considered during pruning, but also the directionality with other vectors. To vectorize a kernel, the kernel must be larger than 1×1. This is because the 1×1 kernel is treated as a scalar with only one component. For kernels larger than 1×1, we can treat the kernel as a vector by treating the elements of the kernel as components of the vector. For example, in the case of a 3×3 kernel, it has 9 elements, which can be expressed as a 9-dimensional vector through vectorization. Here, we express the i-th kernel of the q-th filter of the p-th layer as $K_{q,i}^{(p)}$, and simply express it as K_i when expressed in one filter. Similarly, the q-th filter of the p-th layer is expressed as $F_q^{(p)}$, and when expressing it for one filter, it is simply expressed as F. Then, as shown in Fig.3(a), the i-th vectorized kernel can be expressed as $\overrightarrow{K}_i$, and the filter, which is a set of kernels, is expressed as $F = \{\overrightarrow{K}_0, \overrightarrow{K}_1, \ldots, \overrightarrow{K}_{n-1}\}$. In the end, $\overrightarrow{F}$, which is a filter vector of F, can be obtained by adding all $\overrightarrow{K}$ s, and this process is expressed as follows:

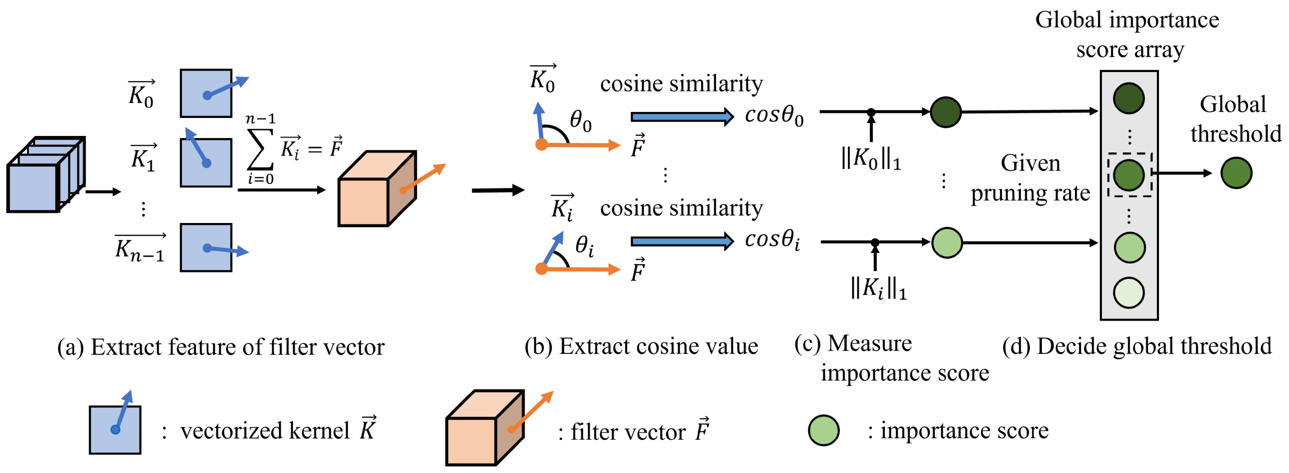
$$\overrightarrow{F} = \sum_{i=0}^{n-1} \overrightarrow{K}_i. \tag{1}$$

$\overrightarrow{F}$ serves as a reference point during kernel pruning. In other words, we aim to keep the properties of $\overrightarrow{F}$ as much as possible during kernel pruning. Therefore, through comparison between $\overrightarrow{K}_i$ and $\overrightarrow{F}$, pruning is performed to minimize changes in the direction and size of $\overrightarrow{F}$. In this paper, the importance of the kernel is determined by considering two factors and stored in S, the importance score of kernels. The importance score of the i-th kernel of the q-th filter of the p-th layer can be expressed as $S_{p,q,i}$, but when expressed for one filter, it is expressed as S_i for simplicity.

First, we determine the importance of the kernel based on the L1 norm. Even if a vector with a small L1 norm is removed, it does not significantly affect the size of $\overrightarrow{F}$ and does not significantly affect the direction of the vector. Second, we determine the importance by reflecting the angle between $\overrightarrow{F}$ and $\overrightarrow{K}_i$ (i.e., θ_i). The orthogonal component of $\overrightarrow{K}_i$ to $\overrightarrow{F}$ interferes with maintaining the direction of $\overrightarrow{F}$. Therefore, in order to give a penalty for the orthogonal component of $\overrightarrow{K}_i$ to $\overrightarrow{F}$, a cosine value is obtained through cosine similarity as shown in Fig.3(b), and it is reflected when calculating the importance score of kernels in (2). This importance scoring process is expressed as follows:

$$S_{p,q,i} = \|K_i\|_1 \times \{\alpha + |cos\theta_i| \times (1 - \alpha)\} \tag{2}$$

where $\|K_i\|_1$ represents the L1 norm as a symbol, and α is a hyperparameter that determines the reflection ratio between the L1 norm and the cosine value. $S_{p,q,i}$ is stored in the global importance score array A_{global}. This $S_{p,q,i}$ is used not only when selecting the kernel to be removed within the filter, but also used to select the layer pruning rate and the global threshold for the pruning rate of each layer. In addition, in the case of 1×1 convolution, since the vectorized kernel-based importance scoring cannot be used, L1 norm-based kernel importance scoring is used independently of



$\overrightarrow{K_0}$
$\overrightarrow{K_1}$
$\overrightarrow{K_{n-1}}$
$\sum_{i=0}^{n-1} \overrightarrow{K_i} = \vec{F}$
cosine similarity
$cos\theta_0$
θ_0
$\vec{F}$
$\overrightarrow{K_i}$
$cos\theta_i$
θ_i
$\|K_0\|_1$
$\|K_i\|_1$
Global importance score array
Global threshold
Given pruning rate
(a) Extract feature of filter vector
(b) Extract cosine value
(c) Measure importance score
(d) Decide global threshold
: vectorized kernel $\vec{K}$
: filter vector $\vec{F}$
: importance score

FIGURE 3. **The process of selecting a global threshold in vectorized kernel pruning. In (a), a vectorized kernel, $\overrightarrow{K}$, is created by vectorizing the kernels in the filter, and the filter vector, $\overrightarrow{F}$, is created by adding all $\overrightarrow{K}$s in the filter. (b) derives $cos\theta_i$ through cosine similarity of $\overrightarrow{K}_i$ and $\overrightarrow{F}$. (c) measures the importance score using the L1 norm and cosine value of the kernel and stores this score in the global importance score array. This process is repeated until importance scores are collected from all kernels in the pruning target. In (d), the importance score corresponding to the given pruning rate in the global importance score array is selected as the global threshold.**

3×3 convolution, and this importance score is used when selecting the global threshold, layer pruning rate, and kernel to be removed in a filter. It should be noted that 1×1 convolution and 3×3 convolution have the same given pruning rate but independent global thresholds.

C. PRUNING PROCESS

Based on the vectorized kernel pruning, we perform pruning in three stages according to a special rule to minimize performance degradation due to the implementation of structured kernel pruning, by reflecting the importance of the layer. First, through the process of Fig.3, a global threshold for the pruning rate of each layer is selected to implement global pruning that reflects the importance of each layer. In detail, the global importance score, S, of all kernels through the process in Fig.3 is sorted in size order and stored in the global importance score array, A_{global}. Assuming that the size of A_{global} is C and the given pruning rate is R_g, the method for obtaining the global threshold T is expressed as follows:

$$T = A_{global}[\lceil C \times R_g \rceil] \tag{3}$$

where $\lceil$ and $\rceil$ are ceiling symbols, and they serve to ceil decimal values because only integers are possible for index. Second, as shown in Fig.4(a), we store the importance scores of the kernels from all filters in a layer in the layer importance score array A_{layer}. Subsequently, as shown in Fig.4(b), through comparison of the global threshold and layer importance score, we divide the number of $S_{p,q,i}$ below T by the total number of kernels in the layer to determine the pruning rate of the layer, $R_l^{(p)}$. Then, for the implementation of structured kernel pruning, kernels are removed by $R_l^{(p)}$ from all filters in one layer as shown in Fig. 4(c). At this time, the kernel to be removed from the filter is selected by comparing the size of S_i. Finally, after creating an unstructured kernel-pruned network by inserting zero values into the kernel to be removed, fine-tuning is performed until the performance is restored. In the third step, as shown in Fig.4(d), we structurally remove the kernels filled with zero values and store the retained kernels and the indexes of the retained kernels. The 1×1 convolution also proceeds with pruning, in the same way, using the importance score obtained by the L1 norm.

It is noteworthy that kernel-pruned weights cannot be operated with general convolution [36]. Because the number of kernels in the filter and the number of input channels in the filter are the same in general convolution, after performing convolution operation in a one-to-one correspondence, they are added to each other to make the output of the filter. However, if kernel pruning is applied, the number of kernels in the filter is less than the number of input channels. In this case, one-to-one correspondence between the kernel and the input channel is not possible, and convolution operation must be performed by designating the input channel corresponding to the retained kernel. To support this operation, it is necessary to separately store the kernel index of the kernel-pruned weight. Therefore, we add the kernel index set to the set storing the index of the retained kernel, and in order to save it, a new parameter is inevitably added in addition to the weight parameter.

To minimize the addition of hardware resources owing to the introduction of new parameters, this study proposes the lossless compression method, which is presented in Fig. 5. Before applying this method, the kernel index set stores the index of the retained kernel as 32 bits. Therefore, if the pruning rate is low, the number of parameters increased by the retained kernel index set may exceed the number of parameters in removed kernels. In addition, in the case of a 1×1 convolution layer in which only one parameter is

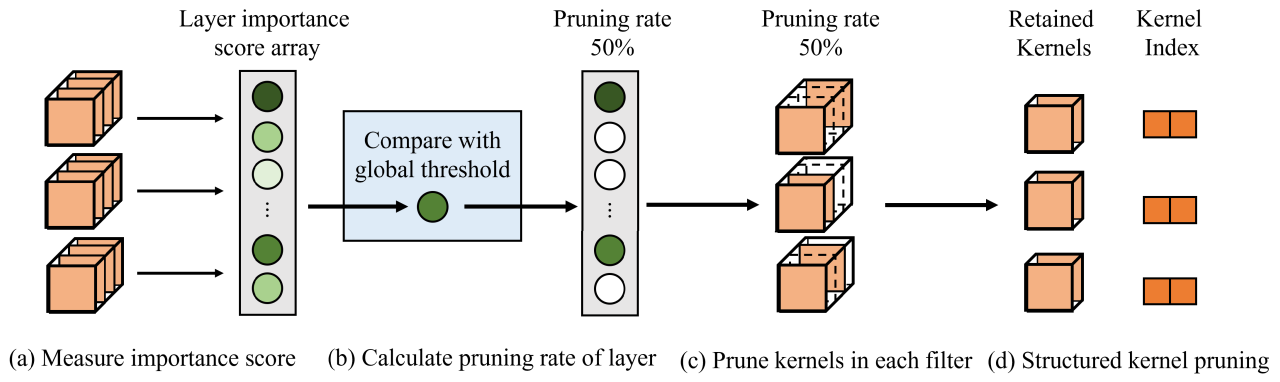

Layer importance score array
Pruning rate 50%
Compare with global threshold
Pruning rate 50%
Retained Kernels
Kernel Index
(a) Measure importance score
(b) Calculate pruning rate of layer
(c) Prune kernels in each filter
(d) Structured kernel pruning

FIGURE 4. Pruning process in each layer. (a) measures the importance score in the kernel of each filter and stores this score in the layer importance score array. (b) compares the global threshold and the layer importance score to calculate the pruning rate of the corresponding layer. (c) fills the kernels below the calculated pruning rate with zero values. (d) structurally removes the kernels filled with zero values and stores the retained kernels and the indexes of the retained kernels.

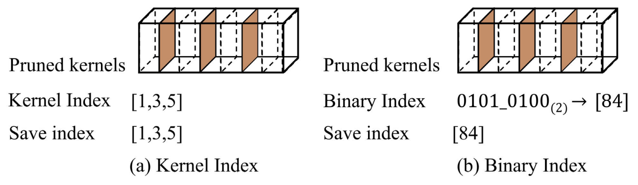


FIGURE 5. Binary index compression method. In (a), all retained indexes should be stored, but in (b), it is possible to store indexes together without loss by compressing them to binary numbers.

present in the kernel, the effect of the addition of the retained kernel index set is even more catastrophic. To alleviate this issue, we apply a lossless compression method as illustrated in Fig. 5. In Fig. 5(a), it is assumed that among the eight kernels, three kernels, [1,3,5], are retained and the kernel index set for the retained kernel is stored as 32 bits. We use the binary index to store not only the index of the retained kernel but also that of the removed kernel. In detail, the index of the removed kernel is converted to 0, and the index of the retained kernel is converted to 1 and arranged in binary numbers in the order of the kernel. When the index of the retained kernel is [1,3,5] as shown in Fig. 5, the index of the eight kernels can be expressed as a binary index of $0101\_0100_{(2)}$. Subsequently, it is converted to [84], which is an 8-bit unsigned number to be stored. Because the size of the kernel index set is maintained regardless of the pruning rate, it is possible to achieve a high level of parameter reduction and weight size reduction by the binary index, even if the pruning rate is low.

D. KERNEL INDEX CONVOLUTION

In this study, we propose a kernel index convolution that matches the kernel of the kernel-pruned weight with the input channels by using the previously stored kernel index set in the proposed kernel pruning process. As shown in Fig. 6, the kernel-pruned weight is different from the number of input channels and the number of kernels in the filter, and it is impossible to know to which input channels the retained kernel corresponds. On the other hand, in the proposed kernel index convolution, since the combination of the retained kernel and the corresponding input channels is stored in the kernel index set in advance, the retained kernel and the corresponding input channels can be known, so that the convolution operation can be performed correctly. As a result, by performing convolution corresponding to the input channels using the index of the retained kernel from the kernel index set, it is possible to generate the output of the kernel index convolution as follows:

$$K_i' * I[idx[i]] = O \tag{4}$$

where idx is the index set of the retained kernel, I is the input channel, O is the output channel, and K' is the retained kernel.

IV. EXPERIMENTS

A. EXPERIMENTAL ENVIRONMENT

- Fine-tuning Environments: All networks are tested in the PyTorch framework, and all weights are pruned using pretrained weights. Stochastic gradient descent (SGD) and a momentum of 0.9 are applied. In Table 1, learning rates of 1e-2, and 1e-3 are sequentially applied without decay. Meanwhile, in Table 2, learning rates of 5e-2, 1e-2, and 1e-3 are sequentially applied without decay. After applying unstructured kernel pruning for fine-tuning, the actual weights are pruned via structured kernel pruning.

- Kernel index convolution environments: The conventional convolution and kernel index convolution implemented in CUDA are connected to the PyTorch framework model through CuPy. The experiment is conducted on a TITAN Xp GPU.

- FLOP reduction and pruned rate: In this study, FLOPs reduction is an indicator of the decreased number of FLOPs in the network after pruning compared with before pruning.

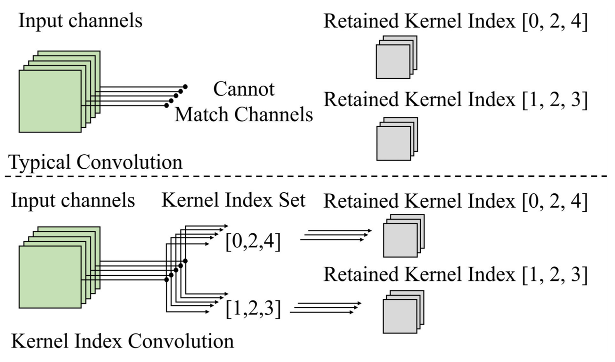

Input channels
Retained Kernel Index [0, 2, 4]
Cannot Match Channels
Retained Kernel Index [1, 2, 3]
Typical Convolution
Input channels
Kernel Index Set
Retained Kernel Index [0, 2, 4]
[0,2,4]
Retained Kernel Index [1, 2, 3]
[1,2,3]
Kernel Index Convolution

FIGURE 6. Kernel index convolution process. In typical convolution, input channels and kernel-pruned weights cannot be matched. However, in the kernel index convolution, the input channel and the retained kernel can be mapped and calculated through the kernel index set.

FLOPs reduction (FR) is calculated as follows:

$$FR = 100\left(1 - \frac{NM' + \sum_{i=0}^{L} F^{(i)}K'^{(i)}(k^{(i)})^2H^{(i)}W^{(i)}}{NM + \sum_{i=0}^{L} F^{(i)}K^{(i)}(k^{(i)})^2H^{(i)}W^{(i)}}\right) \tag{5}$$

where $F^{(i)}$ is the number of filters in the i-th layer, $K^{(i)}$ is the number of kernels in the filter in the i-th layer, $k^{(i)}$ is the size of the kernel in the i-th layer, $K'^{(i)}$ is the number of retained kernels in the filter of the i-th layer after pruning, $H^{(i)}$ is the height of the output of the i-th layer, $W^{(i)}$ is the width of the output of the i-th layer, N is the number of input nodes in the fully connected layer, M is the number of output nodes, and M' is the number of weights connecting the output nodes to the retained input nodes after pruning. In addition, the parameter reduction (PR) is an indicator of the extent to which the network parameters after pruning have decreased compared with the network parameters before pruning, calculated as follows:

$$PR = 100\left(1 - \frac{NM' + \sum_{i=0}^{L} F^{(i)}K'^{(i)}(k^{(i)})^2 + B^{(i)}}{NM + \sum_{i=0}^{L} F^{(i)}K^{(i)}(k^{(i)})^2 + B^{(i)}}\right) \tag{6}$$

where $B^{(i)}$ is the number of scale factors in the BN of the i-th layer.

B. PRUNING RESULTS

To demonstrate that V-SKP is effective in the CIFAR-10 dataset [37], in Table 1, the proposed method is compared with state-of-the-art (SOTA) structured pruning methods used in ResNet-56 and ResNet-110. The proposed V-SKP presents two results according to the FLOPs reduction ratio in each network (i.e., Ours1 & Ours2), and results not provided in the previous studies are indicated by '-'. As a comparative study, TMI-GKP [33] was selected as a priority comparison because it is a structured kernel pruning paper like V-SKP. We also selected KSE [32], which is the most representative of unstructured kernel pruning paper, and PPT-KP [34], which achieved excellent performance as an unstructured kernel pruning research. In addition, a number of representative filter pruning studies, including C-SGD [22], a representative paper that solved the open problem, and ResRep [38], which achieved excellent performance in various networks, were selected for comparison. As can be seen in the experimental results for ResNet-56, although V-SKP (Ours2) has higher FLOP reduction and parameter reduction than KSE [32] by 6.6% and 20.2%, respectively, the accuracy degradation is lower by 0.14%, indicating that V-SKP is superior to KSE [32] which includes the partially structured kernel pruning. In addition, V-SKP (Ours1) achieves higher FLOP reduction and parameter reduction than another structured kernel pruning technique, TMI-GKP [33], by 0.5% and 10%, respectively, while the accuracy performance is also better by 0.45%. The proposed method is slightly better than PPT-KP [34] in terms of the trade-off between accuracy and parameter reduction. It is noteworthy that PPT-KP, an iterative unstructured kernel pruning method that converges the l1 norm of kernels to 0 through sparsity training and then removes the kernel, must be accompanied by fine-tuning after pruning as well as sparsity training that requires considerable computation before pruning. In addition, these processes must be repeated until the target pruning rate is reached, resulting in a very high computational complexity of PPT-KP. On the other hand, since V-SKP does not require a pre-training process for pruning and can be performed only with one-shot pruning, the computational complexity of V-SKP is much lower than that of PPT-KP. OED [19] which achieved the highest FLOPs reduction among comparative studies, caused an additional decrease in accuracy of 1.67%, compared to V-SKP (Ours2), even though OED [19] reduced fewer FLOPs than V-SKP (Ours2). Lastly, V-SKP (Ours2) achieves significantly more FLOPs reduction at a similar accuracy drop compared to ResRep [38], even though ResRep requires additional training. For example, after performing ResRep training for 250 epochs, the filter is removed through a pruning process. On the other hand, the proposed V-SKP performs finetuning for about 80 epochs as a one-shot pruning procedure along with the standard setup at the baseline. Therefore, compared to ResRep, it can be seen that V-SKP has less complexity because it completes the entire procedure through much fewer training epochs. In addition, in the ResRep training process, complexity increases due to the use of penalty gradient, whereas the finetuning process of V-SKP, which is the same as the general training process, does not increase complexity.

In the case of ResNet-110, although V-SKP (Ours2) achieved the highest FLOPs reduction (i.e., 72.8%) and parameter reduction (i.e., 83.1%) compared to previous studies, the accuracy degradation is only 0.02%. In particular, compared to TMI-GKP [33], V-SKP (Ours1) achieves 1.1% of higher FLOP reduction and 10.1% of higher parameter reduction while the accuracy performance is also better by 0.15%. It is noteworthy that V-SKP (Ours1) can achieve FLOPs and parameter reductions of more than 40% and 50%, respectively, with an accuracy improvement of approximately 0.8%.

TABLE 1. Performance comparison on CIFAR-10.

Method	Base Acc	Pruning Type	Pruned Acc. (%)	Acc. Drop (%)	Params. reduction	FLOPs reduction
ResNet-56						
DCP[17]	93.80%	Filter	93.81	-0.01	70.3%	47.1%
HRank[20]	93.26%	Filter	93.17	0.09	42.4%	50.0%
ResRep[38]	93.71%	Filter	93.71	0.00	-	52.9%
ABCPruner[18]	93.26%	Filter	93.23	0.03	54.2%	54.1%
GAL[39]	93.97%	Filter	91.58	2.39	65.9%	60.2%
C-SGD[22]	93.39%	Filter	93.44	-0.05	-	60.9%
OED[19]	93.97%	Filter	92.29	1.68	60.0%	67.7%
TMI-GKP[33]	93.78%	Kernel	94.00	-0.22	43.5%	43.2%
KSE[32]	93.03%	Kernel	92.88	0.15	57.8%	60.0%
PPT-KP[34]	93.39%	Kernel	93.34	0.05	77.2%	68.9%
V-SKP (Ours1)	93.39%	Kernel	94.27	-0.67	53.5%	43.7%
V-SKP (Ours2)	93.39%	Kernel	93.38	0.01	78.0%	66.6%
ResNet110						
FPGM[40]	93.68%	Filter	93.74	-0.06	40.1%	47.4%
FPDC[41]	93.82%	Filter	93.78	0.04	48.8%	48.2%
GAL[39]	94.10%	Filter	92.74	1.36	44.8%	48.5%
OED[19]	94.10%	Filter	93.60	0.50	48.8%	54.1%
HRank[20]	93.50%	Filter	93.36	0.14	59.2%	58.2%
ResRep[38]	94.64%	Filter	93.62	0.02	-	58.2%
C-SGD[22]	94.38%	Filter	94.54	-0.16	-	60.9%
ABCPruner[18]	93.50%	Filter	93.58	-0.08	67.4%	65.0%
TMI-GKP[33]	94.26%	Kernel	94.90	-0.64	43.5%	43.3%
V-SKP (Ours1)	93.68%	Kernel	94.47	-0.79	53.6%	44.4%
V-SKP (Ours2)	93.68%	Kernel	93.66	0.02	83.1%	72.8%

Next, to demonstrate that V-SKP is also effective in the ImageNet ILSVRC-2012 dataset [35], in Table 2, we compare the proposed method with several SOTA structured pruning methods on ResNet-50. We achieve the highest FLOPs reduction only with a similar level of accuracy reduction as in the existing SOTA studies. The top-1 accuracy reduction of LFPC [24] is 1.69%, which is 0.79% lower than that of V-SKP (Ours2) (*i.e.*, 0.90%) despite the FLOP reduction of LFPC being 60.8%, which is 6.3% lower than that of V-SKP (Ours2) (*i.e.*, 67.1%). Another SOTA method, CC [25], shows a parameter reduction of 58.6% and a FLOP reduction of 62.7% with a top-1 accuracy reduction of 1.61%. Although those numbers are 20.9% and 2.4% lower than those of V-SKP (Ours2), respectively, the accuracy of V-SKP (Ours2) is better by 0.71% than CC [25]. In addition, compared to the structured kernel pruning method (*i.e.*, TMI-GKP [33], V-SKP (Ours1) achieves 1.1% and 11.5% of higher FLOP and parameter reductions, respectively, while the accuracy performance is also better by 0.8%. It is also noteworthy that V-SKP (Ours1) can achieve FLOPs and parameter reductions of more than 30% and 40%, respectively, with an accuracy improvement of approximately 0.2% even in the ImageNet dataset. It is noteworthy that TMI-GKP, which structurally removes kernels at the same position from all filters in the group by forming similar filter groups and utilizing group convolution, concatenates each reconstructed input channel according to the order of the filter group. In this process, the number of combined input channels becomes larger than the number of original input channels, and consequently, the memory requirement for the required input increases. On the other hand, V-SKP does not increase the memory requirement of the input channel because it reflects the characteristics of each filter and removes the kernel using kernel index convolution so that the characteristics are maintained as much as possible.

C. ABLATION STUDIES: WEIGHT SIZE, LATENCY, AND COMPATIBILITY EVALUATION

As shown in Table 3, when the binary index, which constitutes the lossless compression method, is used, the weight size is significantly reduced. Because the networks on the CIFAR-10 dataset have a high pruning rate, the weight size is significantly reduced only by the kernel index set (*i.e.*, without using the binary index). However, if the binary index is applied to the kernel index set, then the weight size can be further compressed. In the ImageNet dataset environment, the difference between kernel index and binary index compression becomes significant since the pruning rate is lower than that of CIFAR-10. The experimental result from ResNet-50 shows that when the kernel index set is used, 62.5% of the weight size is decreased, whereas when the binary index is used, 71.4% of the weight size is decreased.

Because the existing framework does not support kernel index convolution, baseline convolution and kernel index

TABLE 2. Performance comparison on imageNet ILSVRC-2012.

Method	Top 1/Top 5 Base Acc. (%)	Pruning Type	Top 1/Top 5 Pruned Acc. (%)	Acc. Drop (%)	Params. reduction	FLOPs reduction
			ResNet-50			
DCP[17]	76.01 / 92.93	Filter	74.95 / 92.32	1.06 / 0.61	51.5%	55.6%
ABCPruner[18]	76.01 / 92.96	Filter	73.52 / 91.51	2.49 / 1.45	56.6%	56.0%
OED[19]	76.15 / 92.87	Filter	74.35 / 91.81	1.80 / 1.06	23.5%	26.9%
HRank[20]	76.15 / 92.87	Filter	74.98 / 92.33	1.17 / 0.54	36.7%	43.8%
LEGR[21]	76.15 / 92.90	Filter	75.30 / 92.40	0.80 / 0.50	-	53.0%
C-SGD[22]	75.33 / 92.56	Filter	74.54 / 92.09	0.79 / 0.47	-	55.8%
ThiNet[23]	72.88 / 91.14	Filter	71.01 / 90.02	1.87 / 1.12	51.6%	55.8%
LFPC[24]	76.15 / 92.87	Filter	74.46 / 92.04	1.69 / 0.83	-	60.8%
CC[25]	76.15 / 92.87	Filter	74.54 / 92.25	1.61 / 1.01	58.6%	62.7%
TMI-GKP[33]	76.15 / -	Kernel	75.53 / -	0.62 / -	33.2%	33.7%
V-SKP (Ours1)	76.13 / 92.86	Kernel	76.33 / 93.00	-0.20 / -0.14	44.7%	34.8%
V-SKP (Ours2)	76.13 / 92.86	Kernel	75.23 / 92.45	0.90 / 0.41	79.5%	67.1%

TABLE 3. Weight Size Comparison Results by the Proposed Method in Various Models and Datasets.

Model	Dataset	Baseline (MB)	Kernel Index (MB)	Binary Index (MB)
ResNet-56	CIFAR-10	3.36	0.95	0.88
ResNet-110		6.80	1.41	1.41
ResNet-50	ImageNet	97.78	36.63	27.97

TABLE 4. Latency Comparison Results by the Proposed Method.

Model	Dataset	Pruning Method	Origin Latency	Pruned Latency	AR
ResNet-56	CIFAR-10	V-SKP(Ours2)	8.0ms	4.7ms	1.70×
ResNet-110			15.7ms	8.3ms	1.89×
ResNet-50	ImageNet	V-SKP(Ours2)	66.7ms	24.5ms	2.72×
		C-SGD[22]		32.2ms	2.07×

convolution are implemented using the CUDA code. Subsequently, an experiment is conducted by linking CuPy with PyTorch. "Origin Latency" in Table 4 refers to the time when inference is performed in the conventional convolution implemented using CUDA code, and "Kernel Index Latency" refers to kernel index convolution (also using CUDA code), where the inference time is measured using the kernel-pruned weight. The presented latency time is the result of inferring one image on each network pruned by V-SKP (Our2), and we present the average value of the inference result of 10,000 images. AR denotes the acceleration rate. Compared with the conventional convolution operation, kernel index convolution includes indexing the input channel via the kernel index set. However, as shown in Table 4, reducing the number of convolutions by skipping input channels via the proposed indexing has advantages, even considering the increase in latency. In the case of the CIFAR-10 dataset, the kernel index latency is accelerated by 1.70× and 1.89× compared with the origin latency in ResNet-56 and ResNet-110, respectively. In addition, ResNet-50 tested in the ImageNet dataset environment shows that V-SKP achieves an acceleration effect of 2.72× accelerations, which is 0.65× higher than C-SGD [22]. This means that the increase in computation complexity due to the input channel indexing process in V-SKP is negligible compared to the reduction in computational complexity and execution time due to pruning accompanying the indexing process.

Next, we analyze the effectiveness of the three stages pruning process of V-SKP explained in Section III-C. In the first stage of V-SKP, the vectorized kernel pruning equation is used to find the global threshold to determine the pruning rate of each layer, and in the second stage of V-SKP, the importance of each kernel is compared using the equation of vectorized kernel pruning. Therefore, we performed additional ablation studies according to the proposed technique at each stage. The experiments in Table 5 compare the results of training ResNet-50 on the ImageNet Dataset for 70 epochs. In the first case, the proposed technique (*i.e.*, Stage 1) is applied only to the global threshold, and the importance of each kernel is configured through a naive method based on the l1-norm. On the other hand, in the second case, the global threshold is calculated using a naive approach based on the l1-norm, and the importance of each kernel is processed through the proposed method (*i.e.*, Stage 2). Table 5 shows that each method of stage 1 and stage 2 causes a top-1 accuracy drop of 1.27% and 2.02%, respectively, compared to the baseline. However, when the two methods are used together, synergy is achieved, making it possible to alleviate even a top-1 accuracy drop of 0.9%. Lastly, it should be noted that if the proposed third stage is not applied, calculations through kernel index convolution are impossible and the actual acceleration effect presented in Table 4 cannot be achieved.

To verify the compatibility of V-SKP with various optimizers, Table 6 shows additional experimental results by changing the fine-tuning of V-SKP (*i.e.*, SGD) to Adam

TABLE 5. Ablation Study for Each Stage of the Proposed Pruning Method.

Method	Top 1/Top 5 Base Acc. (%)	Top 1/Top 5 Pruned Acc. (%)	Acc. Drop (%)	Params. Reduction	FLOPs Reduction
Proposed global threshold + L1-norm-based importance	76.13 / 92.86	74.86 / 92.29	1.27 / 0.57	79.5%	66.3%
L1-norm-based global threshold + Proposed importance		74.11 / 91.98	2.02 / 0.88	79.5%	67.1%
V-SKP (Ours)		75.23 / 92.45	0.90 / 0.41	79.5%	67.1%

TABLE 6. Comparison of Fine-tuning Performance on Cifar-10 according to Various Optimizers.

Model	Optimizer	Base Acc.	Pruned Acc.	Acc. Drop	Params. Reduction	FLOPs Reduction
ResNet-56	SGD	93.39%	93.38%	0.01%	78.0%	66.6%
	Adam[42]		93.37%	0.02%		
	RMSprop[43]		93.25%	0.14%		
ResNet-110	SGD	93.68%	93.66%	0.02%	83.1%	72.8%
	Adam[42]		94.10%	-0.42%		
	RMSprop[43]		93.90%	-0.22%		

[42] and RMSprop [43]. Experimental results show that V-SKP is compatible with any optimizer and can achieve better experimental results than existing pruning studies, even if there is a slight difference in accuracy drop. As a result, V-SKP enables considerable network lightweighting and acceleration with only a small decrease in accuracy, regardless of the optimizer.

V. CONCLUSION

The existing filter pruning method affords a high FLOP reduction rate but has difficulties with application to a complicated network, and the existing kernel pruning methods to compensate for this are difficult to implement via structured pruning; therefore, actual acceleration cannot be achieved in GPUs. To address this problem, this study proposed the novel structured kernel pruning method (i.e., V-SKP) for maintaining the 4D array weight structure and kernel index convolution for supporting the kernel-pruned weights. Consequently, the proposed method cannot only achieve a high parameter reduction rate and FLOP reduction rate, achieving the actual acceleration effect on GPUs. It is noteworthy that the kernel index convolution proposed in this paper is implemented through CUDA code, but due to the limitations of CUDA code, it is less optimized and slower than the implementation on Pytorch. Therefore, there is a need to additionally perform optimization studies on CUDA for practical use of kernel index convolution. Additionally, since vision transformers (ViTs) are being actively researched in the field of computer vision, based on this study, it is expected that future research will be conducted to expand and apply the structured kernel pruning method to the field of ViTs.

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. (CVPR)*, Jun. 2016, pp. 770–778.

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

[3] 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/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 502–511.

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

[5] S. I. Lee and H. Kim, "GaussianMask: Uncertainty-aware instance segmentation based on Gaussian modeling," in *Proc. 26th Int. Conf. Pattern Recognit. (ICPR)*, Aug. 2022, pp. 3851–3857.

[6] Z. Xing, S. Zhao, W. Guo, F. Meng, X. Guo, S. Wang, and H. He, "Coal resources under carbon peak: Segmentation of massive laser point clouds for coal mining in underground dusty environments using integrated graph deep learning model," *Energy*, vol. 285, Dec. 2023, Art. no. 128771. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360544223021655

[7] 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 Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1861–1873, Aug. 2019.

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

[9] N. J. Kim and H. Kim, "FP-AGL: Filter pruning with adaptive gradient learning for accelerating deep convolutional neural networks," *IEEE Trans. Multimedia*, early access, Jul. 11, 2022, doi: 10.1109/TMM.2022.3189496.

[10] E. Camci, M. Gupta, M. Wu, and J. Lin, "QLP: Deep Q-learning for pruning deep neural networks," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 10, pp. 6488–6501, Oct. 2022.

[11] 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.

[12] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2018, *arXiv:1803.03635*.

[13] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[14] A. S. Morcos, H. Yu, M. Paganini, and Y. Tian, "One ticket to win them all: Generalizing lottery ticket initializations across datasets and optimizers," 2019, *arXiv:1906.02773*.

[15] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, B. Catanzaro, and W. J. Dally, "DSD: Dense-sparse-dense training for deep neural networks," 2016, *arXiv:1607.04381*.

[16] X. Ding, G. Ding, X. Zhou, Y. Guo, J. Han, and J. Liu, "Global sparse momentum SGD for pruning very deep neural networks," 2019, *arXiv:1909.12778*.

[17] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–12.

[18] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," 2020, *arXiv:2001.08565*.

[19] Z. Wang, S. Lin, J. Xie, and Y. Lin, "Pruning blocks for CNN compression and acceleration via online ensemble distillation," *IEEE Access*, vol. 7, pp. 175703–175716, 2019.

[20] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "HRank: Filter pruning using high-rank feature map," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1529–1538.

[21] T.-W. Chin, R. Ding, C. Zhang, and D. Marculescu, "Towards efficient model compression via learned global ranking," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1518–1528.

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

[23] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 5058–5066.

[24] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang, "Learning filter pruning criteria for deep convolutional neural networks acceleration," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2006–2015.

[25] Y. Li, S. Lin, J. Liu, Q. Ye, M. Wang, F. Chao, F. Yang, J. Ma, Q. Tian, and R. Ji, "Towards compact CNNs via collaborative compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 6438–6447.

[26] C. Zhao, Y. Zhang, and B. Ni, "Exploiting channel similarity for network pruning," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 33, no. 9, pp. 5049–5061, Sep. 2023.

[27] 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.

[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[29] J. Zhu and J. Pei, "Progressive kernel pruning with saliency mapping of input-output channels," *Neurocomputing*, vol. 467, pp. 360–378, Jan. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231221014879

[30] 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. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–12.

[31] C. Lin, Z. Zhong, W. Wei, and J. Yan, "Synaptic strength for convolutional neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–10.

[32] Y. Li, S. Lin, B. Zhang, J. Liu, D. Doermann, Y. Wu, F. Huang, and R. Ji, "Exploiting kernel sparsity and entropy for interpretable CNN compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2800–2809.

[33] S. Zhong, G. Zhang, N. Huang, and S. Xu, "Revisit kernel pruning with lottery regulated grouped convolutions," in *Proc. Int. Conf. Learn. Represent.*, 2022, pp. 1–23. [Online]. Available: https://openreview.net/forum?id=LdEhiMG9WLO

[34] K. Koo and H. Kim, "PPT-KP: Pruning point training-based kernel pruning for deep convolutional neural networks," in *Proc. IEEE 5th Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2023, pp. 1–5.

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[36] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[37] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.

[38] X. Ding, T. Hao, J. Tan, J. Liu, J. Han, Y. Guo, and G. Ding, "ResRep: Lossless CNN pruning via decoupling remembering and forgetting," 2020, *arXiv:2007.03260*.

[39] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann, "Towards optimal structured CNN pruning via generative adversarial learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2790–2799.

[40] 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/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4340–4349.

[41] Y. Zuo, B. Chen, T. Shi, and M. Sun, "Filter pruning without damaging networks capacity," *IEEE Access*, vol. 8, pp. 90924–90930, 2020.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[43] G. Hinton. *Lecture 6A Overview of Mini-Batch Gradient Descent*. Accessed: Oct. 25, 2023. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

KWANGHYUN KOO (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical and information engineering from the Seoul National University of Science and Technology, Seoul, South Korea, in 2020 and 2022, respectively, where he is currently pursuing the Ph.D. degree in electrical and information engineering. His current research interests include 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, South Korea, in 2009, 2011, and 2015, respectively. From 2015 to 2018, he was a BK Assistant Professor with the BK21 Creative Research Engineer Development for IT, Seoul National University. In 2018, he joined the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, where he is currently an Associate Professor. His current research interests include algorithms, computer architecture, memory, and SoC design for low-complexity multimedia applications and deep neural networks.