REPrune: Channel Pruning via Kernel Representative Selection

Mincheol Park[1,3], **Dongjin Kim**[2,3], **Cheonjun Park**[1], **Yuna Park**[3],
Gyeong Eun Gong[4], **Won Woo Ro**[1], **and Suhyun Kim**[3]*
[1]Yonsei University, Republic of Korea
[2]Korea University, Republic of Korea
[3]Korea Institute of Science and Technology, Republic of Korea
[4]Hyundai MOBIS, Republic of Korea
{mincheol.park, cheonjun.park, wro}@yonsei.ac.kr, npclinic3@korea.ac.kr
{qkrdbsdk0427, dr.suhyun_kim}@gmail.com, gegong@mobis.co.kr

Abstract

Channel pruning is widely accepted to accelerate modern convolutional neural networks (CNNs). The resulting pruned model benefits from its immediate deployment on general-purpose software and hardware resources. However, its large pruning granularity, specifically at the unit of a convolution filter, often leads to undesirable accuracy drops due to the inflexibility of deciding how and where to introduce sparsity to the CNNs. In this paper, we propose REPrune, a novel channel pruning technique that emulates kernel pruning, fully exploiting the finer but structured granularity. REPrune identifies similar kernels within each channel using agglomerative clustering. Then, it selects filters that maximize the incorporation of kernel representatives while optimizing the maximum cluster coverage problem. By integrating with a simultaneous training-pruning paradigm, REPrune promotes efficient, progressive pruning throughout training CNNs, avoiding the conventional train-prune-finetune sequence. Experimental results highlight that REPrune performs better in computer vision tasks than existing methods, effectively achieving a balance between acceleration ratio and performance retention.

Introduction

The growing utilization of convolutional layers in modern CNN architectures presents substantial challenges for deploying these models on low-power devices. As convolution layers account for over 90% of the total computation volume, CNNs necessitate frequent memory accesses to handle weights and feature maps, resulting in an increase in hardware energy consumption (Chen, Emer, and Sze 2016). To address these challenges, network pruning is being widely explored as a viable solution for model compression, which aims to deploy CNNs on energy-constrained devices.

Channel pruning (You et al. 2019; Chin et al. 2020; Sui et al. 2021; Hou et al. 2022) in network pruning techniques stands out as a practical approach. It discards redundant channels—each corresponding to a convolution filter—in the original CNNs, leading to a dense, narrow, and memory-efficient architecture. Consequently, this pruned sub-model is readily deployable on general-purpose hardware, circumventing the need for extra software optimizations (Han, Mao, and Dally 2016). Moreover, channel pruning enhances pruning efficiency by adopting a concurrent training-pruning pipeline. This method entails the incremental pruning of low-ranked channels in training time.

However, pruning a channel (a filter unit), which exhibits larger granularity, encounters greater challenges in preserving accuracy than the method with smaller granularity (Park et al. 2023). This larger granularity often results in undesirable accuracy drops due to the inflexibility in how and where to introduce sparsity into the original model (Zhong et al. 2022). Hence, an approach with a bit of fine-grained pruning granularity, such as kernel pruning (Ma et al. 2020; Niu et al. 2020) illustrated in Fig. 1, has been explored within the structured pruning domain to overcome these limitations.

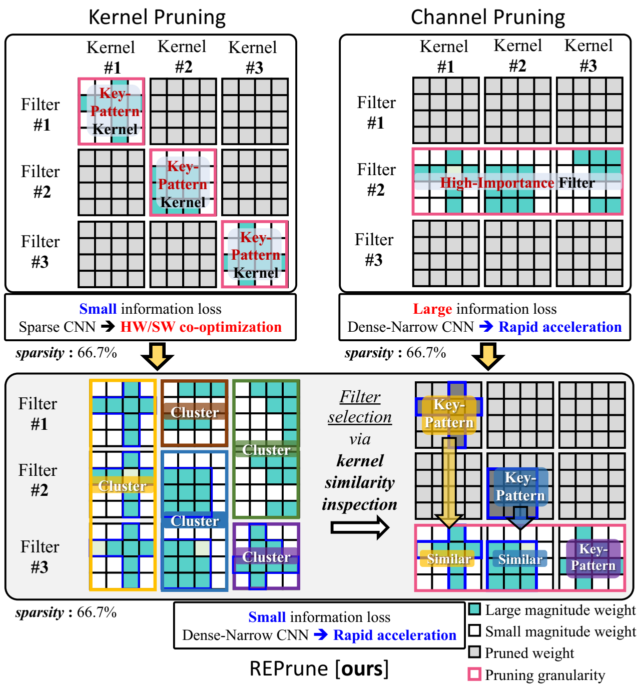

Figure 1: To accelerate CNN and minimize its information loss simultaneously, REPrune intends to select filters associated with patterned key kernels targeted by kernel pruning.

*Corresponding author

This paper focuses on such a kernel, a slice of the 3D tensor of a convolution filter, offering a more fine-grained granularity than an entire filter. Unlike filters, whose dimensions vary a lot according to the layer depth, kernels in modern CNN architectures have exact dimensions, typically 3×3 or 1×1, across all layers. This consistency allows for the fair application of similarity decision criteria. However, naively pruning non-critical kernels (Niu et al. 2020; Zhong et al. 2022) makes CNNs no longer dense. These sparse models require the support of code regeneration or load-balancing techniques to ensure parallel processing so as to accelerate inference (Niu et al. 2020; Park et al. 2023).

Our goal is to develop a channel pruning technique that implies fine-grained kernel inspection. We intend to identify similar kernels and leave one between them. Fig.1 illustrates an example of critical diagonal kernels being selected via kernel pruning. These kernels may also display high similarity with others in each channel, resulting in clusters (Li et al. 2019) as shown in Fig.1. By fully exploiting the clusters, we can obtain the effect of leaving diagonal kernels by selecting the bottom filter, even if it is not kernel pruning. It is worth noting that this selection is not always feasible, but this way is essential to leave the best kernel to produce a densely narrow CNN immediately. In this manner, we can finally represent channel pruning emulating kernel pruning; this is the first attempt to the best of our knowledge.

This paper proposes a novel channel pruning technique entitled "REPrune." REPrune selects filters incorporating the most representative kernels, as depicted in Fig. 2. REPrune starts with agglomerative clustering on a kernel set that generates features of the same input channel. Agglomerative clustering provides a consistent hierarchical linkage. By the linkage sequence, we can get an extent of similarity among kernels belonging to the same cluster. This consistency allows for introducing linkage cut-offs to break the clustering process when the similarity exceeds a certain degree while the clustering process intertwines with a progressive pruning framework (Hou et al. 2022). More specifically, these cut-offs act as linkage thresholds, determined by the layer-wise target channel sparsity inherent in this framework to obtain per-channel clusters. Then, REPrune selects a filter that maximizes the coverage for kernel representatives from all clusters in each layer while optimizing our proposed maximum coverage problem. Our comprehensive evaluation demonstrates that REPrune performs better in computer vision tasks than previous methods.

Our contributions can be summarized as three-fold:

- We present REPrune, a channel pruning method that identifies similar kernels based on clustering and selects filters covering their representatives to achieve immediate model acceleration on general-purpose hardware.
- We embed REPrune within a concurrent training-pruning framework, enabling efficiently pruned model derivation in just one training phase, circumventing the traditional train-prune-finetune procedure.
- REPrune also emulates kernel pruning attributes, achieving a high acceleration ratio with well-maintaining performance in computer vision tasks.

Related Work

Channel pruning Numerous techniques have been developed to retain a minimal, influential set of filters by identifying essential input channels during the training phase of CNNs. Within this domain, training-based methods (Liu et al. 2017; He, Zhang, and Sun 2017; You et al. 2019; Li et al. 2020b) strive to identify critical filters by imposing LASSO or group LASSO penalties on the channels. However, this regularization falls short of enforcing the exact zeroing of filters. To induce complete zeros for filters, certain studies (Ye et al. 2018; Lin et al. 2019) resort to a proxy problem with ISTA (Beck and Teboulle 2009). Other importance-based approaches try to find optimal channels through ranking (Lin et al. 2020), filter norm (He et al. 2018, 2019), or mutual independence (Sui et al. 2021), necessitating an additional fine-tuning process. To fully automate the pruning procedure, sampling-based methods (Kang and Han 2020; Gao et al. 2020) approximate the channel removal process. These methods automate the selection process of channels using differentiable parameters. In contrast, REPrune, without relying on differentiable parameters, simplifies the automated generation of pruned models through a progressive channel pruning paradigm (Hou et al. 2022).

Clustering-based pruning Traditional techniques (Duggal et al. 2019; Lee et al. 2020) exploit differences in filter distributions across layers (He et al. 2019) to identify similarities between channels. These methods group similar filters or representations, mapping them to a latent space, and retain only the essential ones. Recent studies (Chang et al. 2022; Wu et al. 2022) during the training phase of CNNs cluster analogous feature maps to group the corresponding channels. Their aim is to ascertain the optimal channel count for each layer, which aids in designing a pruned model. REPrune uses clustering that is nothing too novel. In other words, REPrune departs from the filter clustering but performs kernel clustering in each input channel to ultimately prune filters. This is a distinct aspect of REPrune.

Kernel pruning Most existing methods (Ma et al. 2020; Niu et al. 2020) retain specific patterns within convolution kernels, subsequently discarding entire connections associated with the least essential kernels. To improve kernel pruning, previous studies (Yu et al. 2017; Zhang et al. 2022) have initiated the process by grouping filters based on their similarity. A recent work (Zhong et al. 2022) combines the Lottery Ticket Hypothesis (Frankle and Carbin 2019) to group cohesive filters early on, then optimizes kernel selection through their specialized weight selection process. However, such methodologies tend to produce sparse CNNs, which demand dedicated solutions (Park et al. 2023) for efficient execution. Diverging from this challenge, REPrune focuses on selecting filters that best represent critical kernels, circumventing the need for extra post-optimization.

Methodology

Prerequisite We denote the number of input and output channels of the l-th convolutional layer as n_l and n_{l+1}, respectively, with $k_h \times k_w$ representing the kernel dimension

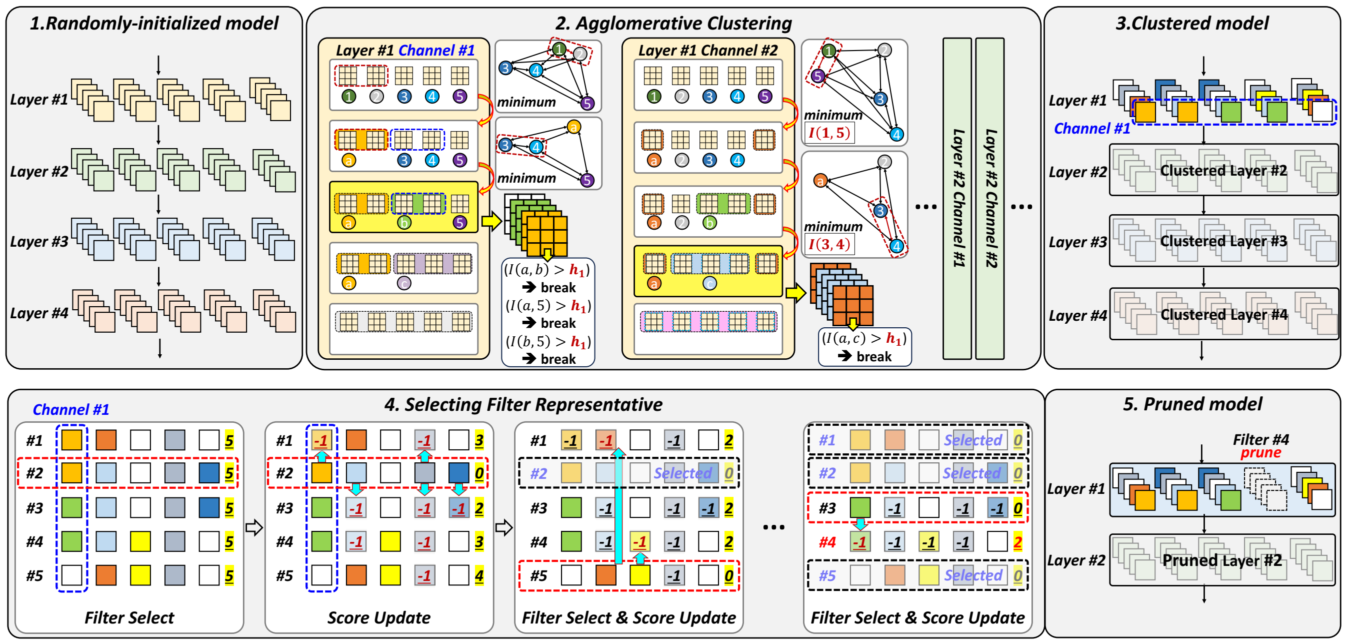

Figure 2: An overview of the REPrune methodology for identifying redundant kernels and selecting filters. Every channel performs agglomerative clustering on its corresponding kernel set in each layer. Once clusters are formed in accordance with the target channel sparsity, our proposed solver for the MCP starts its greedy filter selection until the target number of channels is satisfied. This solver selects a filter that includes a representative kernel from each grouped cluster per channel.

of the l-th convolution filters. A filter set for the l-th layer are represented as $\mathcal{F}^l \in \mathbb{R}^{n_{l+1}\times n_l\times k_h\times k_w}$. We define the j-th set of kernels of $\mathcal{F}^l$ as $\mathcal{K}^l_j \in \mathbb{R}^{n_{l+1}\times k_h\times k_w}$, where j refers to the index of an input channel. Each individual kernel is denoted by $\kappa^l_{i,j} \in \mathbb{R}^{k_h\times k_w}$, resulting in $\mathcal{K}^l_j = \{\kappa^l_{i,j} : \forall i \in \mathcal{I}\}$. Lastly, we define $\mathcal{I} = \{1, ..., n_{l+1}\}$ and $\mathcal{J} = \{1, ..., n_l\}$.

Preliminary: Agglomerative Clustering

Agglomerative clustering acts on 'bottom-up' hierarchical clustering by starting with singleton clusters (i.e., separated kernels) and then iteratively merging two clusters into one until only one cluster remains.

Let a *cluster* set after the agglomerative clustering repeats c-th merging process for the j-th kernel set $\mathcal{K}^l_j$ as $\mathcal{AC}_c(\mathcal{K}^l_j)$. This process yields a sequence of intermediate cluster sets $\mathcal{AC}_0(\mathcal{K}^l_j), ..., \mathcal{AC}_{n_{l+1}-1}(\mathcal{K}^l_j)$ where $\mathcal{AC}_0(\mathcal{K}^l_j) = \{\{\kappa^l_{i,j}\} : \forall\kappa^l_{i,j} \in \mathcal{K}^l_j\}$ denotes the initial set of individual kernels, and $\mathcal{AC}_{n_{l+1}-1}(\mathcal{K}^l_j) = \{\mathcal{K}^l_j\}$ is the root cluster that incorporates all kernels in $\mathcal{K}^l_j$. This clustering process can be typically encapsulated in the following recurrence relationship:

$$\mathcal{AC}_c(\mathcal{K}^l_j) = \mathcal{AC}_{c-1}(\mathcal{K}^l_j) \setminus \{A_c, B_c\} \cup \{A_c \cup B_c\}, \quad (1)$$

where A_c and B_c denotes two arbitrary clusters from $\mathcal{AC}_{c-1}(\mathcal{K}^l_j)$, which are the closest in the merging step c. The distance between two clusters can be measured using the single linkage, complete linkage, average linkage, and Ward's linkage distance (Witten and Frank 2002).

This paper exploits Ward's method (Ward Jr 1963). First, it prioritizes that kernels within two clusters are generally not too dispersed in merging clusters (Murtagh and Legendre 2014). In other words, Ward's method can calculate the dissimilarity of all combinations of the two clusters at each merging step c. Thus, at each merging step c, a pair of clusters to be grouped can be identified based on the maximum similarity value, which represents the smallest relative distance compared to other distances. Ward's linkage distance defines $I(A_c, B_c) = \Delta(A_c \cup B_c) - \Delta(A_c) - \Delta(B_c)$ where $\Delta(\cdot)$ is the sum of squared error. Then, the distance between two clusters is defined as follows:

$$I(A_c, B_c) = \frac{|A_c|\,|B_c|}{|A_c| + |B_c|}\|\mathbf{m}_{A_c} - \mathbf{m}_{B_c}\|_2. \quad (2)$$

where $\mathbf{m}_{A_c}$ and $\mathbf{m}_{B_c}$ are the centroid of each cluster and $|\cdot|$ is the number of kernels in them. Hence, Ward's linkage distance starts at zero when every kernel is a singleton cluster and grows as we merge clusters hierarchically. Second, Ward's method ensures this growth monotonically (Milligan 1979). Let $d(c; \mathcal{K}^l_j)$, the minimum of Ward's linkage distances, be the linkage objective function in the c-th linkage step: $d(c; \mathcal{K}^l_j) = \min_{(A_c,B_c)\in\mathcal{AC}_{c-1}(\mathcal{K}^l_j)} I(A_c, B_c)$, s.t., $A_c \neq B_c, \forall c \in \{1, ..., n_{l+1} - 1\}$ and $d(0; \mathcal{K}^l_j) = 0$. Then, $d(c-1; \mathcal{K}^l_j) \leq d(c; \mathcal{K}^l_j)$. The non-decreasing property of Ward's linkage distance leads to a consistent linkage order, although the clustering repeats. By the monotonicity of Ward's method, agglomerative clustering can even perform until it reaches a cut-off height h as follows:

$$\mathbf{1}_c(h; \mathcal{K}^l_j) = \begin{cases} \mathcal{AC}_c(\mathcal{K}^l_j), & h \geq d(c; \mathcal{K}^l_j), \\ \mathcal{AC}_{c-1}(\mathcal{K}^l_j), & \text{otherwise}, \end{cases} \quad (3)$$

where $\mathbf{1}_c(\cdot)$ is a linkage control function to perform agglomerative clustering in c-th merging step. The monotonicity of Ward's method allows for a direct mapping of Ward's linkage distance value to the height h, which can then be used as a distance value. Therefore, Eq.3 can be used to generate available clusters until $d(c; \mathcal{K}_j^l)$ does not exceed h as the desired cut-off parameter.

The following section will introduce how we set h as the linkage cut-off to determine per-channel clusters, and then REPrune makes the per-channel clusters through Eq. 3.

Foundation of Clusters Per Channel

This section introduces how to set a layer-specific linkage cut-off, termed h_l, and demonstrates the process to produce clusters per channel using h_l in each l-th convolutional layer. Instead of applying a uniform h across all layers (Duggal et al. 2019), our strategy employs a unique h_l to break the clustering process when necessary in each layer. This ensures that the distances between newly formed clusters remain out of a threshold so as to preserve an acceptable degree of dispersion in each layer accordingly.

Our method begins with agglomerative clustering on each kernel set $\mathcal{K}_j^l$. Given the l-th layer's channel sparsity $s^l \in [0, 1]$[1], kernel sets continue to cluster until $\lceil (1 - s^l) n_{l+1} \rceil$ clusters form[2]. In other words, agglomerative clustering needs to repeat $\tilde{n}_{l+1}$ (where $\tilde{n}_{l+1} = \lceil s^l n_{l+1} \rceil$) merger steps. In this paper, we make the cluster set per channel as $\mathcal{AC}_{\tilde{n}_{l+1}}(\mathcal{K}_1^l)$ up to $\mathcal{AC}_{\tilde{n}_{l+1}}(\mathcal{K}_{n_l}^l)$.

At this merging step $\tilde{n}_{l+1}$, we collect Ward's linkage distances, denoted as $d(\tilde{n}_{l+1}; \mathcal{K}_1^l)$ through $d(\tilde{n}_{l+1}; \mathcal{K}_{n_l}^l)$. Note that kernel distributions can vary significantly across channels (Li et al. 2019). This means some channels may generate distances that are too close, reflecting the cohesiveness of the kernels, while others may yield greater distances due to the diversity of the kernels.

The diversity of kernels can lead to variations in the number of clusters for each channel. Furthermore, a smaller cut-off height tends to increase this number of clusters, which indicates a high preservation of kernel representation. To both preserve the representation of each channel and ensure channel sparsity simultaneously, we set h_l to the maximum value among the collected distances as follows:

$$h_l = \max_{j \in \mathcal{J}} d(\tilde{n}_{l+1}; \mathcal{K}_j^l), \quad l \in \mathcal{L}. \tag{4}$$

The linkage cut-off h_l serves as a pivotal parameter in identifying cluster sets in each channel based on the channel sparsity s^l. This linkage cut-off h_l aids in choosing suitable cluster set $\mathcal{AC}^*(\mathcal{K}_j^l)$ between $\mathcal{AC}_{\tilde{n}_{l+1}-1}(\mathcal{K}_j^l)$ and $\mathcal{AC}_{\tilde{n}_{l+1}}(\mathcal{K}_j^l)$, using the control function defined in Eq. 3:

$$\mathcal{AC}^*(\mathcal{K}_j^l) = \mathbf{1}_{\tilde{n}_{l+1}}(h_l; \mathcal{K}_j^l), \quad j \in \mathcal{J}, l \in \mathcal{L}. \tag{5}$$

Each cluster, derived from Eq. 5, indicates a group of similar kernels, any of which can act as a representative within its cluster. The following section provides insight into selecting a filter that optimally surrounds these representatives.

[1]The process for obtaining s^l will be outlined in detail within the overview of the entire pipeline in this section.

Filter Selection via Maximum Cluster Coverage

This section aims to identify a subset of the filter set $\mathcal{F}^l$ to yield the maximum coverage for kernel representatives across all clusters within each l-th convolutional layer.

We frame this objective within the Maximum Coverage Problem (MCP). In this formulation, each kernel in a filter is directly mapped to a distinct cluster. Further, a *coverage score*, either 0 or 1, is allocated to each kernel. This score indicates whether the cluster corresponding to a given kernel has already been represented as filters are selected. Therefore, our primary strategy for optimizing the MCP involves prioritizing the selection of filters that maximize the sum of these coverage scores. This way, we approximate an entire representation of all clusters within the reserved filters.

To define our MCP, we introduce a set of clusters U^l from the l-th convolutional layer. This set, denoted as $U^l = \{\mathcal{AC}^*(\mathcal{K}_1^l); \cdots; \mathcal{AC}^*(\mathcal{K}_{n_l}^l)\}$, represents the clusters that need to be covered. Given a filter set $\mathcal{F}^l$, each j-th kernel $\kappa_{i,j}^l$ of a filter corresponds to a cluster in the set $\mathcal{AC}^*(\mathcal{K}_j^l)$, where $\forall i \in \mathcal{I}$ and $\forall j \in \mathcal{J}$. This way, we can view each filter as a subset of U^l. This perspective leads to the subsequent set cover problem, which aims to approximate U^l using an optimal set $\tilde{\mathcal{F}}^l$ that contains only the necessary filters.

The objective of our proposed MCP can be cast as a minimization problem that involves a subset of filters, denoted as $\tilde{\mathcal{F}}^l \subset \mathcal{F}^l$, where $\tilde{\mathcal{F}}^l \in \mathbb{R}^{\lceil (1-s^l) n_{l+1} \rceil \times n_l \times k_h \times k_w}$ represents the group of filters in the l-th layer that are not pruned:

$$\min \left(|U^l| - \sum_{\mathcal{F}_r^l \in \tilde{\mathcal{F}}^l} \sum_{\kappa_{r,j}^l \in \mathcal{F}_r^l} \sum_{j \in \mathcal{J}} S(\kappa_{r,j}^l) \right), \tag{6}$$
$$\text{s.t.}, S(\kappa_{r,j}^l) \in \{0, 1\}, \ |\tilde{\mathcal{F}}^l| = \lceil (1 - s^l) n_{l+1} \rceil,$$

where $\mathcal{F}_r^l$ is a selected filter, and $|U^l| = \sum_{j=1}^{n_l} |AC^*(\mathcal{K}_j^l)|$ is the sum of optimal coverage scores. *Each kernel in $\mathcal{F}_r^l$ initially has a coverage score, denoted by $S(\kappa_{r,j}^l)$, of one. This score transitions to zero if the cluster from $\mathcal{AC}^*(\mathcal{K}_j^l)$, to which $\kappa_{r,j}^l$ maps, is already covered.*

As depicted in Fig. 2, we propose a greedy algorithm to optimize Eq. 6. This algorithm selects filters encompassing the maximum number of uncovered clusters as follows:

$$i = \text{argmax}_{i \in \mathcal{I}} \sum\nolimits_{j \in \mathcal{J}} S(\kappa_{i,j}^l), \quad \forall \kappa_{i,j}^l \in \mathcal{F}_i^l. \tag{7}$$

There may be several candidate filters, $\mathcal{F}_i^l, \mathcal{F}_{i'}^l, ..., \mathcal{F}_{i''}^l$ (where $i \neq i' \neq \cdots \neq i''$), that share the maximum, identical coverage scores. Given that any filter among these candidates could be equally representative, we resort to random sampling to select a filter $\mathcal{F}_r^l$ from them.

Upon the selection and inclusion of a filter $\mathcal{F}_r^l$ into $\tilde{\mathcal{F}}^l$, the coverage scores of each kernel in the remaining filters from $\mathcal{F}^l$ are updated accordingly. This update continues in the repetition of the procedure specified in Eq. 7 until a total of $\lceil (1 - s^l) n_{l+1} \rceil$ filters have been selected. This thorough process is detailed in Alg.1.

[2]The ceiling function is equal to $\lceil x \rceil = \min\{n \in \mathbb{Z} : n \geq x\}$.

Algorithm 1: Channel Selection in REPrune

```
Input: Filter set $\mathcal{F}^l = \{\mathcal{K}_j^l : j \in \mathcal{J}\}, \forall l \in \mathcal{L}$
Output: Non-pruned filter set $\tilde{\mathcal{F}}^l$, $\forall l \in \mathcal{L}$[1]
Given: $\mathcal{AC}$; Target channel sparsity $s^l$, $\forall l \in \mathcal{L}$
 1: for layer $l \in \mathcal{L}$ do
 2:     for channel $j$ from 1 to $n_l$ do
 3:         $\mathcal{AC}_c(\mathcal{K}_j^l)$ until merging step $c$ becomes $\tilde{n}_{l+1}$
 4:         Get $d(\tilde{n}_{l+1}; \mathcal{K}_j^l)$ and then obtain a cut-off $h_l$
 5:         Set clusters per channel $\mathcal{AC}^*(\mathcal{K}_j^l)$        ▷ Eq. 5
 6:     end for
 7:     Define the optimal cluster coverage $U^l$
 8:     Initialize $\tilde{\mathcal{F}}^l$ as an empty queue
 9:     while $|\tilde{\mathcal{F}}^l| < \lceil (1-s^l) n_{l+1} \rceil$ do
10:         Select candidate filters in $\mathcal{F}^l$        ▷ Eq. 7
11:         Sample $\mathcal{F}_r^l$ from the candidates and add to $\tilde{\mathcal{F}}^l$
12:         Update coverage scores of remaining kernels
13:     end while
14: end for
```

Algorithm 2: Overview of REPrune

```
Input: A CNN model $\mathcal{M}$ with convolution filters $\{\mathcal{F}^l : \forall l \in \mathcal{L}\}$; agglomerative clustering algorithm $\mathcal{AC}$; total number of channel pruning epochs $T_{prune}$; pruning interval epoch $\Delta T$; global channel sparsity $\bar{s}$; dataset $\mathcal{D}$
Output: A pruned model with filters $\{\tilde{\mathcal{F}}^l : \forall l \in \mathcal{L}\}$
 1: Initialize $\mathcal{M}$
 2: $t \leftarrow 1$        ▷ $t$ denotes epoch
 3: while $\mathcal{M}$ is not converged do
 4:     Draw mini-batch samples from $\mathcal{D}$
 5:     Perform gradient descent on $\mathcal{M}$
 6:     if $t$ mod $\Delta T = 0$ and $t \leq T_{prune}$ then
 7:         Compute channel sparsity $\{s^l : \forall l \in \mathcal{L}\}$   ▷ Eq.9
 8:         Execute channel selection via REPrune[2] ▷ Alg.1
 9:         Perform the channel regrowth process
10:     end if
11:     $t \leftarrow t + 1$
12: end while
```

Complete Pipeline of REPrune

We introduce REPrune, an efficient pruning pipeline enabling concurrent channel exploration within a CNN during training. This approach is motivated by prior research (Liu et al. 2017; Ye et al. 2018; Zhao et al. 2019), which leverages the trainable scaling factors (γ) in Batch Normalization layers to assess channel importance during the training process. We define a set of scaling factors across all layers as $\Gamma = \{\gamma_i^l : (i,l) \in (\mathcal{I}, \mathcal{L})\}$. A quantile function, $Q : [0,1] \longmapsto \mathbb{R}$, is applied to Γ using a global channel sparsity ratio $\bar{s} \in [0,1)$. This function determines a threshold γ^*=$Q(\bar{s}; \Gamma)$, which is used to prune non-critical channels:

$$Q(\bar{s}; \Gamma) = \inf\{\gamma_i^l \in \Gamma : F(\gamma_i^l) \geq \bar{s}, (i,l) \in (\mathcal{I}, \mathcal{L})\}, \quad (8)$$

where $F(\cdot)$ denotes the cumulative distribution function. The layer-wise channel sparsity s^l can then be derived using this threshold as follows:

$$s^l = \frac{1}{|\mathcal{I}|} \sum\nolimits_{i \in \mathcal{I}} \mathbb{1}(\gamma_i^l \leq \gamma^*), \quad l \in \mathcal{L}, \quad (9)$$

where the indicator function, $\mathbb{1}(\cdot)$, returns 0 if $\gamma_j^l \leq \gamma^*$ is satisfied and 1 otherwise. This allows channels in each layer to be automatically pruned if their corresponding scaling factors are below γ^*.

In some cases, Eq.9 may result in s^l being equal to one. This indicates that all channels in the l-th layer must be removed at specific training iterations, which consequently prohibits agglomerative clustering. We incorporate the *channel regrowth strategy* (Hou et al. 2022) to tackle this problem. This strategy allows for restoring some channels pruned in previous training iterations. In light of this auxiliary approach, REPrune seamlessly functions without interruption during the training process.

This paper introduces an inexpensive framework to serve REPrune while simultaneously training the CNN model. The following section will demonstrate empirical evaluations of how this proposed method, fully summarized in Alg. 2, surpasses the performance of previous techniques.

Experiment

This section extensively evaluates REPrune across image recognition and object detection tasks. We delve into case studies investigating the cluster coverage ratio during our proposed MCP optimization. Furthermore, we analyze the impacts of agglomerative clustering with other monotonic distances. We also present REPrune's computational efficiency in the training-pruning time and the image throughput at test time on AI computing devices.

Datasets and models We evaluate image recognition on CIFAR-10 and ImageNet (Deng et al. 2009) datasets and object detection on COCO-2017 (Lin et al. 2014). For image recognition, we use various ResNets (He et al. 2016), while for object detection, we employ SSD300 (Liu et al. 2016).

Evaluation settings This paper evaluates the effectiveness of REPrune using the PyTorch framework, building on the generic training strategy from DeepLearningExample (Shen et al. 2022). While evaluations are conducted on NVIDIA RTX A6000 with 8 GPUs, for the CIFAR-10 dataset, we utilize just a single GPU. Additionally, we employ NVIDIA Jetson TX2 to assess the image throughput of our pruned model with A6000. Comprehensive details regarding training hyper-parameters and pruning strategies for CNNs are available in the Appendix. This paper computes FLOPs by treating both multiplications and additions as a single operation, consistent with the approach (He et al. 2016).

[1]The number of original in-channels n_l of not pruned filters $\tilde{\mathcal{F}}^l$ is not the same as the out-channel $\lceil (1-s^{l-1}) n_l \rceil$ of $\tilde{\mathcal{F}}^{l-1}$. To address this issue, we use a hard pruning method (He et al. 2018) that adjusts the n_l in-channels to match $\lceil (1-s^{l-1}) n_l \rceil$ of $\tilde{\mathcal{F}}^{l-1}$.

[2]When the event where $s^l = 1$ is encountered, indicating the pruning of all channels in l-th convolutional layer, the sub-routine of REPrune is exceptionally skipped.

Method	PT?	FLOPs	Top-1	Epochs
ResNet-18				
Baseline	-	1.81G	69.4	-
PFP (Liebenwein et al. 2020)	✓	1.27G	67.4	270
SCOP-A (Tang et al. 2020)	✓	1.10G	69.1	230
DMCP (Guo et al. 2020)	✗	1.04G	69.0	150
SOSP (Nonnenmacher et al. 2022)	✗	1.20G	68.7	128
REPrune	✗	**1.03G**	**69.2**	**250**
ResNet-34				
Baseline	-	3.6G	73.3	-
GReg-1 (Wang et al. 2021)	✓	2.7G	73.5	180
GReg-2 (Wang et al. 2021)	✓	2.7G	73.6	180
DTP (Li et al. 2023)	✗	2.7G	74.2	180
REPrune	✗	**2.7G**	**74.3**	**250**
GFS (Ye et al. 2020)	✓	2.1G	72.9	240
DMC (Gao et al. 2020)	✓	2.1G	72.6	490
SCOP-A (Tang et al. 2020)	✓	2.1G	72.9	230
SCOP-B (Tang et al. 2020)	✓	2.0G	72.6	230
NPPM (Gao et al. 2021)	✓	2.1G	73.0	390
CHEX (Hou et al. 2022)	✗	2.0G	73.5	250
REPrune	✗	**2.0G**	**73.9**	**250**
ResNet-50				
Baseline	-	4.1G	76.2	-
Hrank (Lin et al. 2020)	✓	2.3G	75.0	570
HALP (Shen et al. 2022)	✓	3.1G	77.2	180
GReg-1 (Wang et al. 2021)	✓	2.7G	76.3	180
SOSP (Nonnenmacher et al. 2022)	✗	2.4G	75.8	128
REPrune	✗	**2.3G**	**77.3**	**250**
SCP (Kang and Han 2020)	✗	1.9G	75.3	200
CHIP (Sui et al. 2021)	✓	2.1G	76.2	180
CafeNet (Su et al. 2021)	✗	2.0G	76.9	300
NPPM (Su et al. 2021)	✓	1.8G	75.9	300
EKG (Lee and Song 2022)	✓	2.2G	76.4	300
HALP (Shen et al. 2022)	✓	2.0G	76.5	180
EKG+BYOL[5] (Lee and Song 2022)	✓	1.8G	76.6	300
DepGraph (Fang et al. 2023)	✓	1.9G	75.8	-
REPrune	✗	**1.8G**	**77.0**	**250**
DMCP (Guo et al. 2020)	✗	1.1G	74.1	150
EagleEye (Li et al. 2020a)	✓	1.0G	74.2	240
ResRep (Ding et al. 2021)	✓	1.5G	75.3	270
GReg-2 (Wang et al. 2021)	✓	1.3G	73.9	180
DSNet (Li et al. 2021)	✓	1.2G	74.6	150
CHIP (Sui et al. 2021)	✓	1.0G	73.3	180
CafeNet (Su et al. 2021)	✗	1.0G	75.3	300
HALP+EagleEye (Shen et al. 2022)	✓	1.2G	74.5	180
SOSP (Nonnenmacher et al. 2022)	✗	1.1G	73.3	128
HALP (Shen et al. 2022)	✓	1.0G	74.3	180
DTP (Li et al. 2023)	✗	1.7G	75.5	180
DTP (Li et al. 2023)	✗	1.3G	74.2	180
REPrune	✗	**1.0G**	**75.7**	**250**

Table 1: Top-1 accuracy comparison with channel pruning methods on the ImageNet dataset. 'PT?' indicates whether a method necessitates pre-training the original CNN. 'Epochs' represent the sum of training and fine-tuning time or the entire training time for methods without pre-training CNN.

Image Recognition

Comparison with channel pruning methods Table 1 shows the performance of REPrune when applied to ResNet-18, ResNet-34, and ResNet-50 on the ImageNet dataset. REPrune demonstrates notable efficiency, with only a slight decrease or increase in accuracy at the smallest FLOPs compared to the baseline. Specifically, the accuracy drop is only 0.2% and 0.5% for ResNet-18 and ResNet-50 models, concerning FLOPs of 1.03G and 1.0G, respectively. Moreover, the accuracy of ResNet-34 improves by 0.6% when its FLOPs reach 2.0G. Beyond this, REPrune's accuracy not only surpasses the achievements of contemporary influential channel pruning techniques but also exhibits standout results, particularly for the 2.7G FLOPs of ResNet-34 and the 2.3G and 1.8G FLOPs of ResNet-50–all of which display about 1.0% improvements in accuracy over their baselines.

Method	PT?	FLOPs reduction	Baseline Top-1	Pruned Top-1	Epochs
ResNet-56 (FLOPs: 127M)					
CUP (Duggal et al. 2019)	✗	52.83%	93.67	93.36	360
LSC (Lee et al. 2020)[6]	✓	55.45%	93.39	93.16	1160
ACP (Chang et al. 2022)	✓	54.42%	93.18	93.39	530
REPrune	✗	**60.38%**	**93.39**	**93.40**	**160**
ResNet-18 (FLOPs: 1.81G)					
CUP (Duggal et al. 2019)	✗	43.09%	69.87	67.37	180
ACP (Chang et al. 2022)	✓	34.17%	70.02	67.82	280
REPrune	✗	**43.09%**	**69.40**	**69.20**	**250**
ResNet-50 (FLOPs: 4.1G)					
CUP (Duggal et al. 2019)	✗	54.63%	75.87	74.34	180
ACP (Chang et al. 2022)	✓	46.82%	75.94	75.53	280
REPrune	✗	**56.09%**	**76.20**	**77.04**	**250**

Table 2: Comparison with clustering-based channel pruning methods, including analysis using ResNet-56 on CIFAR-10.

Method	PT?	FLOPs reduction	Baseline Top-1	Pruned Top-1	Epochs
ResNet-56 (FLOPs: 127M)					
GKP (Zhong et al. 2022)	✗	43.23%	93.78	94.00	300
REPrune	✗	**47.57%**	**93.39**	**94.00**	**160**
ResNet-50 (FLOPs: 4.1G)					
GKP (Zhong et al. 2022)	✗	33.74%	76.15	75.53	90
REPrune	✗	**56.09%**	**76.20**	**77.04**	**250**

Table 3: Comparison with a recent kernel pruning method.

Comparison with clustering-based methods As shown in Table 2, REPrune stands out with FLOPs reduction rates of 60.38% for ResNet-56 and 56.09% for ResNet-50. Each accuracy is compelling, with a slight increase of 0.01% for ResNet-56 and a distinctive gain of 0.84% over the baseline for ResNet-50. At these acceleration rates, REPrune's accuracy of 93.40% (ResNet-56) and 77.04% (ResNet-50) surpass those achieved by prior clustering-based methods, even at their lower acceleration rates.

Comparison with kernel pruning method Table 3 shows that REPrune excels by achieving a FLOPs reduction rate of 56.09% for ResNet-50 on ImageNet. In this FLOPs gain, REPrune even records a 0.84% accuracy improvement to the baseline. This stands out when compared to GKP. On CIFAR-10, REPrune achieves greater FLOPs reduction than GKP while maintaining the same level of performance.

[5]BYOL (Grill et al. 2020) is a self-supervised learning method.
[6]There is no reported accuracy on the ImageNet dataset.

Method	FLOPs reduction	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
SSD300 object detection							
Baseline	0%	25.2	42.7	25.8	7.3	27.1	40.8
DMCP (Guo et al. 2020)	50%	24.1	41.2	24.7	6.7	25.6	39.2
REPrune	**50%**	**25.0**	**42.3**	**25.9**	**7.4**	**26.8**	**40.4**

Table 4: Evaluation of SSD300 with ResNet-50 the backbone on COCO-2017. Performance is assessed using bounding box AP. FLOPs reduction refers solely to the backbone.

Linkage method	Single	Complete	Average	Ward
FLOPs reduction	60.70%	64.04%	63.96%	60.38%
Top-1 accuracy	92.74%	92.67%	92.87%	93.40%

Table 5: Comparison of linkage methods for ResNet-56 on CIFAR-10, using the same global channel sparsity, $\bar{s}$=0.55.

FLOPs reduction	0%	43%	56%	75%
Training time (hours)	43.75	40.62	38.75	35.69

Table 6: Training time comparison for ResNet-50 using the REPrune on ImageNet. The '1 hour' is equivalent to training with 8 RTX A6000 GPUs with 16 worker processes.

Object Detection

Table 4 presents that, even with a FLOPs reduction of 50% in the ResNet-50 backbone, REPrune surpasses DMCP by 0.9 in mAP, a minor decrease of 0.2 compared to the baseline.

Case Study

Impact of other monotonic linkage distance Agglomerative clustering is available with other linkage methods, including single, complete, and average, each of which satisfies monotonicity for bottom-up clustering. As shown in Table 5, when Ward's linkage method is replaced with any of these alternatives, accuracy retention appears limited, especially when imposing the same channel sparsity of 55%. In the context of the FLOPs reduction with the same channel sparsity, we conjecture that using these alternatives in REPrune tends to favor removing channels from the front layers compared to Ward's method. These layers have more massive FLOPs and are more information-sensitive (Li et al. 2017). While each technique reduces a relatively high number of FLOPs, they may also suffer from maintaining accuracy due to removing these sensitive channels.

Cluster coverage rate of selected filters Fig. 3 illustrates how coverage rates change over time, aggregating selected filter scores relative to the sum of optimal coverage scores. Initially, clusters from subsequent layers predominate those from the front layers. As learning continues, REPrune progressively focuses on enlarging the channel sparsity of front layers. This dynamic, when coupled with our greedy algorithm's thorough optimization of the MCP across all layers, results in a steady increase in coverage rates, reducing the potential for coverage failures.

Training-pruning time efficiency Table 6 shows the training (forward and backward) time efficiency of REPrune applied to ResNet-50 on the ImageNet dataset. REPrune results in time savings of 3.13, 5.00, and 8.06 hours for FLOPs reductions of 43%, 56%, and 75%, respectively. This indicates that it can reduce training time effectively in practice.

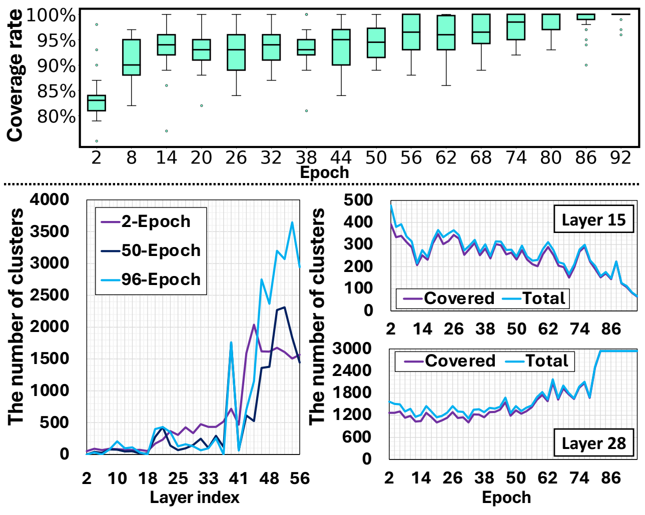

Figure 3: The illustration of coverage rates for ResNet-56 on CIFAR-10 during the optimization of our proposed MCP. Each box contains the coverage rates from all pruned convolutional layers throughout the training epoch.

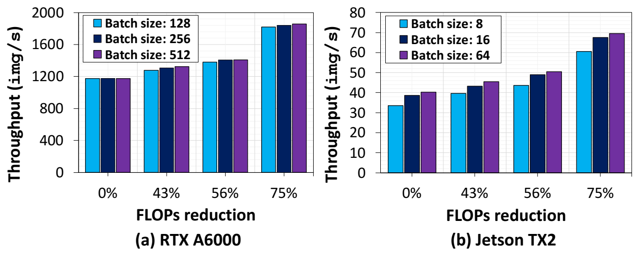

Figure 4: Computing throughput (`img/s`) on image inference using ResNet-50 on the ImageNet validation dataset.

Test-time image throughput Fig. 4 demonstrates the image throughput of pruned ResNet-50 models on both a single RTX A6000 and a low-power Jetson TX2 GPU in realistic acceleration. These models were theoretically reduced by 43%, 56%, and 75% in FLOPs and were executed in various batch sizes. Discrepancies between theoretical and actual performance may arise from I/O delay, frequent context switching, and the number of CUDA and tensor cores.

Conclusion

Channel pruning techniques suffer from large pruning granularity. To overcome this limitation, we introduce REPrune, a new approach that aggressively exploits the similarity between kernels. This method identifies filters that contain representative kernels and maximizes the coverage of kernels in each layer. REPrune takes advantage of both kernel pruning and channel pruning during the training of CNNs, and it can preserve performance even when acceleration rates are high.

Acknowledgements

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-00456, Development of Ultra-high Speech Quality Technology for remote Multi-speaker Conference System), and by the Korea Institute of Science and Technology (KIST) Institutional Program.

References

Beck, A.; and Teboulle, M. 2009. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*.

Chang, J.; Lu, Y.; Xue, P.; Xu, Y.; and Wei, Z. 2022. Automatic channel pruning via clustering and swarm intelligence optimization for CNN. *Applied Intelligence*.

Chen, Y.-H.; Emer, J.; and Sze, V. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News*.

Chin, T.-W.; Ding, R.; Zhang, C.; and Marculescu, D. 2020. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Ding, X.; Hao, T.; Tan, J.; Liu, J.; Han, J.; Guo, Y.; and Ding, G. 2021. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Duggal, R.; Xiao, C.; Vuduc, R. W.; and Sun, J. 2019. Cup: Cluster pruning for compressing deep neural networks. *2021 IEEE International Conference on Big Data*.

Fang, G.; Ma, X.; Song, M.; Mi, M. B.; and Wang, X. 2023. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Frankle, J.; and Carbin, M. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*.

Gao, S.; Huang, F.; Cai, W.; and Huang, H. 2021. Network pruning via performance maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Gao, S.; Huang, F.; Pei, J.; and Huang, H. 2020. Discrete model compression with resource constraint for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Grill, J.-B.; Strub, F.; Altché, F.; Tallec, C.; Richemond, P.; Buchatskaya, E.; Doersch, C.; Avila Pires, B.; Guo, Z.; Gheshlaghi Azar, M.; et al. 2020. Bootstrap your own latent-a new approach to self-supervised learning. In *Advances in Neural Information Processing Systems*.

Guo, S.; Wang, Y.; Li, Q.; and Yan, J. 2020. Dmcp: Differentiable markov channel pruning for neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep compression: compressing deep neural network with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

He, Y.; Kang, G.; Dong, X.; Fu, Y.; and Yang, Y. 2018. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*.

He, Y.; Liu, P.; Wang, Z.; Hu, Z.; and Yang, Y. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Hou, Z.; Qin, M.; Sun, F.; Ma, X.; Yuan, K.; Xu, Y.; Chen, Y.-K.; Jin, R.; Xie, Y.; and Kung, S.-Y. 2022. Chex: Channel exploration for CNN model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Kang, M.; and Han, B. 2020. Operation-aware soft channel pruning using differentiable masks. In *International Conference on Machine Learning*.

Lee, S.; Heo, B.; Ha, J.-W.; and Song, B. C. 2020. Filter pruning and re-initialization via latent space clustering. *IEEE Access*.

Lee, S.; and Song, B. C. 2022. Ensemble knowledge guided sub-network search and fine-tuning for filter pruning. In *Proceedings of the European Conference on Computer Vision*.

Li, B.; Wu, B.; Su, J.; and Wang, G. 2020a. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *Proceedings of the European Conference on Computer Vision*.

Li, C.; Wang, G.; Wang, B.; Liang, X.; Li, Z.; and Chang, X. 2021. Dynamic slimmable network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2017. Pruning filters for efficient convnets. In *International Conference on Learning Representations*.

Li, Y.; Gu, S.; Mayer, C.; Gool, L. V.; and Timofte, R. 2020b. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Li, Y.; Lin, S.; Zhang, B.; Liu, J.; Doermann, D.; Wu, Y.; Huang, F.; and Ji, R. 2019. Exploiting kernel sparsity and entropy for interpretable CNN compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Li, Y.; van Gemert, J. C.; Hoefler, T.; Moons, B.; Eleftheriou, E.; and Verhoef, B.-E. 2023. Differentiable transportation pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Liebenwein, L.; Baykal, C.; Lang, H.; Feldman, D.; and Rus, D. 2020. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*.

Lin, M.; Ji, R.; Wang, Y.; Zhang, Y.; Zhang, B.; Tian, Y.; and Shao, L. 2020. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Lin, S.; Ji, R.; Yan, C.; Zhang, B.; Cao, L.; Ye, Q.; Huang, F.; and Doermann, D. 2019. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft coco: Common objects in context. In *Proceedings of the European Conference on Computer Vision*.

Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; and Berg, A. C. 2016. Ssd: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision*.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Ma, X.; Guo, F.-M.; Niu, W.; Lin, X.; Tang, J.; Ma, K.; Ren, B.; and Wang, Y. 2020. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Milligan, G. W. 1979. Ultrametric hierarchical clustering algorithms. *Psychometrika*.

Murtagh, F.; and Legendre, P. 2014. Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion? *Journal of Classification*.

Niu, W.; Ma, X.; Lin, S.; Wang, S.; Qian, X.; Lin, X.; Wang, Y.; and Ren, B. 2020. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*.

Nonnenmacher, M.; Pfeil, T.; Steinwart, I.; and Reeb, D. 2022. Sosp: Efficiently capturing global correlations by second-order structured pruning. In *International Conference on Learning Representations*.

Park, C.; Park, M.; Oh, H. J.; Kim, M.; Yoon, M. K.; Kim, S.; and Ro, W. W. 2023. Balanced column-wise block pruning for maximizing GPU parallelism. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Shen, M.; Yin, H.; Molchanov, P.; Mao, L.; Liu, J.; and Alvarez, J. M. 2022. Structural pruning via latency-saliency knapsack. In *Advances in Neural Information Processing Systems*.

Su, X.; You, S.; Huang, T.; Wang, F.; Qian, C.; Zhang, C.; and Xu, C. 2021. Locally Free Weight Sharing for Network Width Search. In *International Conference on Learning Representations*.

Sui, Y.; Yin, M.; Xie, Y.; Phan, H.; Aliari Zonouz, S.; and Yuan, B. 2021. Chip: Channel independence-based pruning for compact neural networks. In *Advances in Neural Information Processing Systems*.

Tang, Y.; Wang, Y.; Xu, Y.; Tao, D.; Xu, C.; Xu, C.; and Xu, C. 2020. Scop: Scientific control for reliable neural network pruning. In *Advances in Neural Information Processing Systems*.

Wang, H.; Qin, C.; Zhang, Y.; and Fu, Y. 2021. Neural pruning via growing regularization. In *International Conference on Learning Representations*.

Ward Jr, J. H. 1963. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*.

Witten, I. H.; and Frank, E. 2002. Data mining: Practical machine learning tools and techniques with Java implementations. *ACM Sigmod Record*.

Wu, Z.; Li, F.; Zhu, Y.; Lu, K.; Wu, M.; and Zhang, C. 2022. A filter pruning method of CNN models based on feature maps clustering. *Applied Sciences*.

Ye, J.; Lu, X.; Lin, Z.; and Wang, J. Z. 2018. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*.

Ye, M.; Gong, C.; Nie, L.; Zhou, D.; Klivans, A.; and Liu, Q. 2020. Good subnetworks provably exist: Pruning via greedy forward selection. In *International Conference on Machine Learning*.

You, Z.; Yan, K.; Ye, J.; Ma, M.; and Wang, P. 2019. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*.

Yu, N.; Qiu, S.; Hu, X.; and Li, J. 2017. Accelerating convolutional neural networks by group-wise 2D-filter pruning. *International Joint Conference on Neural Networks*.

Zhang, G.; Xu, S.; Li, J.; and Guo, A. J. X. 2022. Group-based network pruning via nonlinear relationship between convolution filters. *Applied Intelligence*.

Zhao, C.; Ni, B.; Zhang, J.; Zhao, Q.; Zhang, W.; and Tian, Q. 2019. Variational convolutional neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Zhong, S.; Zhang, G.; Huang, N.; and Xu, S. 2022. Revisit kernel pruning with lottery regulated grouped convolutions. In *International Conference on Learning Representations*.