REVISIT KERNEL PRUNING WITH LOTTERY REGULATED GROUPED CONVOLUTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Structured pruning methods which are capable of delivering a densely pruned network are among the most popular techniques in the realm of neural network pruning, where most methods prune the original network at a filter or layer level. Although such methods may provide immediate compression and acceleration benefits, we argue that the blanket removal of an entire filter or layer may result in undesired accuracy loss. In this paper, we revisit the idea of kernel pruning (to only prune one or several $k \times k$ kernels out of a 3D-filter), a heavily overlooked approach under the context of structured pruning due to it will naturally introduce sparsity to filters within the same convolutional layer—thus, making the remaining network no longer dense. We address this problem by proposing a versatile grouped pruning framework where we first cluster filters from each convolutional layer into equal-sized groups, prune the grouped kernels we deem unimportant from each filter group, then permute the remaining filters to form a densely grouped convolutional architecture (which also enables the parallel computing capability) for fine-tuning. Specifically, we consult empirical findings from a series of literature regarding *Lottery Ticket Hypothesis* to determine the optimal clustering scheme per layer, and develop a simple yet cost-efficient greedy approximation algorithm to determine which group kernels to keep within each filter group. Furthermore, we discuss how this pruning framework will potentially work with more advanced unsupervised clustering schemes and inductive biases on weights shifting discovered in the future. Extensive experiments also demonstrate our method often outperforms comparable SOTA methods with lesser data argumentation needed, smaller fine-tuning budget required, and sometimes even with much simpler procedure executed (e.g., one-shot pruning v. iterative pruning, standard fine-tuning v. custom fine-tuning).

1 INTRODUCTION

The applications of convolutional neural networks (CNNs) have demonstrated proven success in various computer vision tasks (Voulodimos et al., 2018). However, with modern CNN architectures being increasingly deeper and wider, over-parameterization has become one of the major challenges of deploying such models to devices with limited computational resources and memory capacity (Frankle & Carbin, 2019). Therefore, the study of network pruning—the technique of removing redundant weights from the originally trained network without significantly sacrificing accuracy—has been an important subject both for practical concerns (Mozer & Smolensky, 1989) and to better understand the properties and mechanisms of neural networks (Arora et al., 2018).

In the realm of CNN pruning, a spectrum of techniques have been maturely studied where the two ends are *structured* pruning and *unstructured* pruning (Mao et al., 2017). Methods from the former end often propose to remove redundant weights in groups while following some geometrical constraints—such as removing a certain filter or layer. The methods from the latter end, on the other hand, prune the network with a more fine-grained view where they evaluate every weight individually. Yet, there are many other methods lie in between of the two ends. Methods that are more "unstructured" are believed to be capable of yielding better accuracy retention with a commensurate amount of parameters pruned, due to having a higher degree of freedom on how and where to introduce sparsity to the original dense network. Empirical findings also support this claim (Liu et al., 2019b; Mao et al., 2017).

Despite having advantages on accuracy retention, the resultant network from methods closer to the unstructured end will be more sparse. Which may not provide actual compression and acceleration without relying on custom-indexing, sparse convolution libraries, or even dedicated hardware devices; thus, limits the deployability of such methods (Mao et al., 2017). Meanwhile, methods closer to the structured end, by preserving a more regulated resultant network, are more likely to be library/hardware-friendly. The most deployable structured pruning method may deliver a dense resultant network and therefore gain immediate compression and acceleration benefits. We denote this kind of pruning method as *densely structured*.

Naturally, many scholars want to develop new methods within the realm of densely structured pruning but with better accuracy retention. The majority of densely structured pruning methods focus on pruning the original dense network at a filter- or layer-level. We argue that the blanket removal of an entire filter or layer may harm the representation power of the network and result in undesired accuracy loss—as removing a filter would consequently remove all feature maps generated by such filter. Even worse, removing a layer would eliminate more feature maps and even face the danger of *layer-collapse*, a phenomenon of having an untrainable pruned network due to premature pruning of an entire layer (Tanaka et al., 2020).

In this paper, we revisit the idea of kernel pruning (to only prune one or several $k \times k$ kernels from a 3D-filter, instead of the entire one) as an alternative and less aggressive pruning approach. We hypothesize that by not removing the entire filter, the representation power of the original network will be better preserved. Although the idea of kernel pruning is nothing too novel (as it is simply a special case of individual weights pruning with 100% of weights of a kernel pruned), it is mostly applied under the context of unstructured pruning or structured pruning methods which may not deliver a dense pruned network (Mao et al., 2017; Ma et al., 2020). This is because a direct implementation of kernel pruning with no constraint would introduce sparsity across the network and therefore making the pruned network no longer dense. We address this problem by proposing a versatile grouped pruning framework, where we:

1. Cluster similar filters from each convolutional layer into a (predefined) number of equal sized filter groups.
2. For each filter group, identify a certain portion of grouped kernels to prune according to the required pruning ratio.
3. Permute the remaining filters to form a densely grouped convolutional architecture according to the number of groups used in step 1.

Like most other post-train grouped pruning methods, we face the challenge of determining which clustering schemes and which importance metrics to use in step 1 and 2 of the above procedure. Upon investigations and experiments, we discovered that a classical clustering scoring system (e.g., the *Silhouette score*) might not capture the better clustering scheme in regard to accuracy retention. Yet many filter importance metrics require sophisticated procedures, which are not computationally friendly or easy to execute when applied at a kernel-level. We address the first challenge by consulting model-generated information—in this case, the empirical findings on *Lottery Ticket Hypothesis* and related literature on weights shifting—to develop a scoring system that identifies the optimal clustering scheme among options per each convolutional layer (Frankle & Carbin, 2019; Renda et al., 2020). For the second challenge, we design a simple and cost-efficient greedy algorithm with multiple restarts to generate multiple candidate kernel selection queues and identify the one queue where the preserved kernels are most "distinctive" from each other yet "similar" to the pruned kernels. The main contributions and advantages of our method are:

- We brought attention to the heavily overlooked approach of kernel pruning under the context of densely structured pruning.
- We developed a framework that is compatible with further-developed/discovered clustering schemes and inductive biases, or more advanced variations upon them. This overcomes one of the major drawbacks of many filter pruning methods: as most of them propose different filter importance metrics that are largely incompatible with each other either in terms of their procedure or computation.
- The resultant network of our method is structured as a densely grouped convolution, which enables parallel computing capability and greatly increases the practical deployability of

our methods: as we can now share the required computation and memory footprint across multiple end-user devices, where most of them have very limited said resources individually (e.g., IoT devices, mobile phones, and wearable technologies).

- Even by just applying well-understood classical mathematical tools, extensive experiments demonstrate our method outperforms comparable SOTA methods (with much more sophisticated procedures) across different networks and datasets by a sensible margin. Additionally, our method often needs less data augmentation, a smaller fine-tuning budget, and it executes much "cleaner" without requiring any custom retraining, special fine-tuning, or iterative cycles—which is rare for approaches relying on *Lottery Ticket Hypothesis*-related studies.

2 Related Work

Many prior arts have explored the possibility of obtaining a smaller model with comparable performance by removing redundant weights (Zhu & Gupta, 2017; Han et al., 2016), filters (Molchanov et al., 2017; Yu et al., 2018; He et al., 2019; Wang et al., 2019a), layers (Wang et al., 2019c; Lin et al., 2019a), image input (Howard et al., 2017; Han et al., 2020), or from all three dimensions (Wang et al., 2021a). It is clear that filter pruning attracts the most attention among all structured pruning approaches.

Our method is inspired by group convolution, a is well convolutional architecture which could be implemented efficiently on common devices (Iandola et al., 2017). Although kernel pruning used together with filter clustering is not a popular trend, we have seen such a combination in Yu et al. (2017). However, the proposed method by Yu requires iterative analysis of many different intermediate feature maps per layer, involves a complex knowledge distillation application during the fine-tuning stage, and lacks comparable experiment results to recent pruning literature. Where our method delivers beyond-SOTA performance with a straightforward one-shot pruning and standard fine-tuning procedure. In addition, our method consults empirical findings on the **lottery ticket hypothesis** and its derived literature regarding weights masking and weights shifting (Frankle & Carbin, 2019; Renda et al., 2020; Zhou et al., 2019). More on this in the Proposed Method section.

3 Proposed Method

3.1 Preliminaries

Assume a convolutional neural network $\boldsymbol{W}$ has L convolutional layers, we denote the $\boldsymbol{W}^\ell$ to be the ℓ-th convolutional layer of $\boldsymbol{W}$ (for $\ell \in \{\mathbb{Z}^+ \mid [1, L]\}$). Therefore, we shall have a 4-D tensor $\boldsymbol{W}^\ell \in \mathbb{R}^{C^\ell_{\text{out}} \times C^\ell_{\text{in}} \times H^\ell \times W^\ell}$ where C^ℓ_{out} represents the number of filters in $\boldsymbol{W}^\ell$ (also known as the number of *output channels* in some literature), C^ℓ_{in} represents the number of kernels per each filter (a.k.a. number of *input channels*), and $H^\ell \times W^\ell$ represents the size of each kernel.

The overall procedure of our method can be mainly divided into four stages: 1) Clustering filters into n equal-sized groups, where the best clustering scheme for each convolutional layer is determined using the *tickets magnitude increase* score derived from prior arts on lottery ticket hypothesis and weight-shifting; 2) Evaluating several candidate-grouped kernel queues generated by the dynamic-programming algorithm with multiple restarts, where the queue with preserved kernels that are most distinctive from each other, yet most similar to the pruned kernels get selected; 3) Permuting the preserved filters to form a grouped convolution architecture with n groups; 4) Fine-tuning the pruned and grouped network with f more epochs to recover accuracy lost from pruning.

3.2 Clustering filters into groups

The first step of our method is to cluster filters from the same convolutional layer into n equal-sized groups. Known that for a layer $\boldsymbol{W}^\ell$ we have C^ℓ_{out} filters, there shall be C^ℓ_{out}/n filters inside each equal-sized group. We denotes $\boldsymbol{F}^\ell_i$ to be the i-th filter in $\boldsymbol{W}^\ell$ (namely, $\boldsymbol{F}^\ell_i = \boldsymbol{W}^\ell_{[i,:,:,:]}$) for $i \in \{\mathbb{Z}^+ \mid [1, C^\ell_{\text{out}}]\}$. Filter clustering is a maturely adopted technique in network pruning since it is a widely accepted assumption that when similar filters are clustered together, the representation

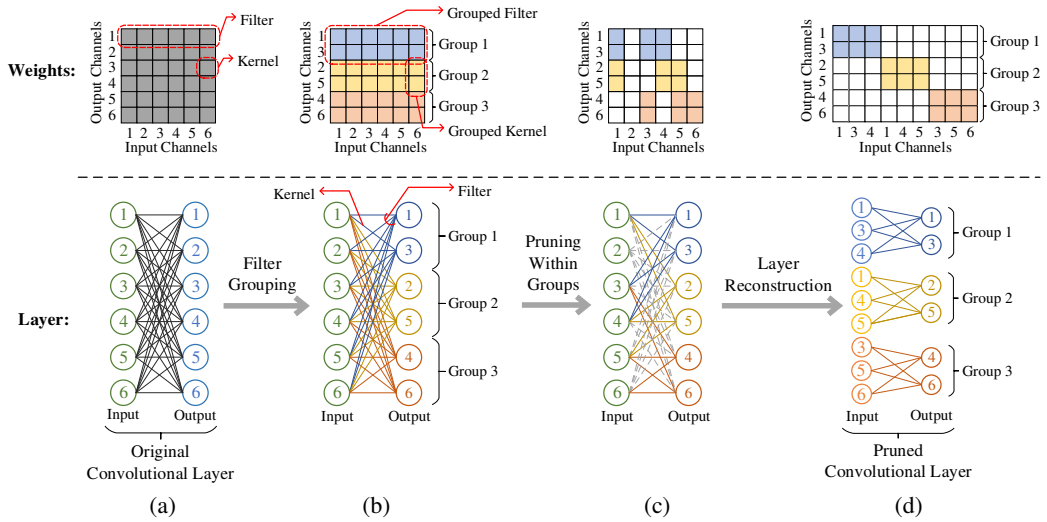

Figure 1: Steps of our method

power of some filters can be covered by the rest of the filters in the same group (which therefore enables the potential of pruning). Additionally, this technique drastically decomposes the scope of the problem, as we may now proceed to evaluate in a group-by-group fashion instead of evaluating all of the filters from $\boldsymbol{W}^\ell$ together.

Many prior arts have developed methods on filter clustering with linearity assumptions (Guo et al., 2020), via retraining with a custom loss function (Wu et al., 2018), or through an iterative process (Yu et al., 2017). We argue that since each filter $\boldsymbol{F}_i^\ell$ is a tensor of $C_{\text{in}}^\ell \times H^\ell \times W^\ell$, considerations regarding non-linearity and high-dimensionality relationships should be added. Therefore, we utilize the following three combinations of proven mathematical tools on dimensionality reduction and clustering in order to cluster filters from each layer into n equal-sized groups.

3.2.1 Available Clustering Schemes

To reduce the dimension of filters in $\boldsymbol{W}^\ell$ and cluster them into n equal-sized groups, we utilize the following three sets of methods: **K-PCA + k-Means**, **Spectral Clustering + k-Means**, and **t-SNE + DBSCAN** (with slight modifications done to obtain the desired cluster size and number). Since all five methods here have been maturely studied, we will omit the introduction of these methods; please refer to section Dimensionality Reduction and Clustering Schemes in Appendix for details.

3.2.2 Decide Optimal Clustering Scheme per Layer with TMI Score

With the three proposed and many other unimplemented sets of clustering schemes, one natural question to ask is how are we suppose to decide the optimal clustering scheme? And at what scale should we make such a decision? The latter question was relatively easier to answer as we already set on clustering filters into groups in a layer-by-layer fashion, so it is reasonable enough to find out the most suitable clustering scheme for each layer. But to address the former question, we need to find a way to quantify the "quality" of proposed clustering results. Mathematicians have provided us with classical tools such as the *Silhouette score* to measure the quality of clusters. However, experiment results suggest such tools can hardly identify the better cluster method under the context of accuracy retention (A.1.1).

We hypothesize that this is because tools like the Silhouette score focus on properties regarding clusters themselves—such as distances between clusters, or cohesion within each cluster—while accuracy retention relays filters importance, or balance between filter groups under the context of the network. Therefore, we decide to introduce model-induced information to help us quantify the quality of different cluster results: where we consult empirical findings from a series of literature regarding *Lottery Ticket Hypothesis* and weights shifting.

Background on Lottery Ticket Hypothesis, Weights Shifting, and Weights Rewinding For a dense convolutional neural network $\boldsymbol{W}$ initialized with weights of $\boldsymbol{W}_0$, the conventional post-train pruning procedure is three steps:

1. **Train** the proposed network for t epochs. Where we denote the network has a weights of $\boldsymbol{W}_k$ at the k-th epoch for $0 \leq k \leq t$.
2. **Prune** the network with weights $\boldsymbol{W}_t$ according to the defined pruning ratio and achieve subnetwork $\boldsymbol{W}_t^{'}$, where $\boldsymbol{W}_t^{'} \subset \boldsymbol{W}_t$.
3. **Fine-tune** the network with $\boldsymbol{W}_t^{'}$ and f more epochs to recover accuracy.

The *Lottery Ticket Hypothesis* claims that instead of fine-tuning upon the subnetwork $\boldsymbol{W}_t^{'}$, there exists a subnetwork $\boldsymbol{W}_k^{'}$ (defined as $\boldsymbol{W}_t^{'}$ but replaced with its counterpart weights from $\boldsymbol{W}_k$) that, when trained in isolation for f more epochs, may match or outperform the test accuracy of fine-tuned $\boldsymbol{W}_t^{'}$ achieved from step 3 above. This $\boldsymbol{W}_k^{'}$ is therefore known as the *winning ticket* (Frankle & Carbin, 2019).

A vast amount of research has been done to demonstrate the existence of winning tickets across different networks and datasets, making the lottery ticket hypothesis one of the most tested inductive biases among neural networks. Scholars have additionally investigated the relationship between the winning ticket's weights and final weights in regard to accuracy retention. Zhou et al. (2019) did one of the most thorough experiments on exploring such relationship by deploying nine different zero *mask criteria* related to initial weights, final weights, and weights shifting during training where they concluded that $||\boldsymbol{W}_t| - |\boldsymbol{W}_0||$, the *magnitude increase* of a network at its initialization and after training, has demonstrated most significant correlation with accuracy retention.

Note that the Zhou et al. (2019) experiments were conducted under the assumption that the winning ticket exists at $\boldsymbol{W}_0$. This is because, for early or even current arts on the lottery ticket hypothesis, scholars have their disagreements on whether to reinitialize the weights of $\boldsymbol{W}_t^{'}$ to their initial values (namely, $k = 0$ for k in $\boldsymbol{W}_k^{'}$) (Frankle & Carbin, 2019), to near initialization ($0 < k \ll t$) (Renda et al., 2020), or just to reset randomly (Liu et al., 2019b). However, with more comprehensive experiments conducted, the findings reveal there are multiple winning tickets that exist within a range of epochs starting from near initialization: such that for any k where $0 < k_1 \leq k \leq k_2 < t$, $\boldsymbol{W}_k^{'}$ can be a winning ticket (with k_1 being close to 0). This technique is referred as *weights rewinding* and we denotes the range of $[k_1, k_2]$ as tickets window (Renda et al., 2020).

Ticket Magnitude Increase scoring system Based on the fact that finding a winning ticket may occur at near initialization, but not necessary exactly at initialization, we slightly modify the *magnitude increase* criteria from Renda et al. (2020) by replacing the initialization weights as ticket weights. Thus, we define the *ticket magnitude increase* score (hereinafter "TMI score") as $||\boldsymbol{W}_t| - |\boldsymbol{W}_k||$ for $0 < k \ll t$. Since we aim to use such scoring system to govern the clustering quality of filter groups, we further expand this scoring system from individual weight to a filter-level. For a filter $\boldsymbol{F}_i^\ell \in \mathbb{R}^{C_{\text{in}}^\ell \times H^\ell \times W^\ell}$, we denote the weight at (h, w) index from the c-th kernel of $\boldsymbol{F}_i^\ell$ as $\boldsymbol{W}_{[i,c,h,w]}^\ell$; so the TMI score for filter $\boldsymbol{F}_i^\ell$ in regard to a wining ticket at the k-th epoch should be defined as the following.

$$\text{TMI}(\boldsymbol{F}_i^\ell, k) = \sum_{c}^{C_{in}^\ell} \sum_{h}^{H^\ell} \sum_{w}^{W^\ell} (||\boldsymbol{W}_{t_{[i,c,h,w]}}^\ell| - |\boldsymbol{W}_{k_{[i,c,h,w]}}^\ell||). \tag{1}$$

Similarly, for a filter group $\boldsymbol{g}^\ell$ of $\{\boldsymbol{F}_i^\ell, \boldsymbol{F}_j^\ell, \dots\}$, its TMI score would be:

$$\text{TMI}(\boldsymbol{g}^\ell, k) = \sum_{g_i \in \boldsymbol{g}} \text{TMI}(\boldsymbol{F}_{g_i}^\ell, k). \tag{2}$$

Qualifying cluster results with TMI scores With the *TMI score* for a filter group defined, the next question to address is what makes a good clustering result in regards to TMI scores? We

already observed that tools like the Silhouette score could not capture the better clustering result in terms of accuracy retention, and we hypothesized that this is due to the lack of attention on **filter importance** and **balance between filter groups**. Since prior art Zhou et al. (2019) has established the relationship between better accuracy retention and weights with larger magnitude increase, we expect such relationship will expand to a filter-level where filters with large TMI scores may help on accuracy retention and therefore be deemed "more important."

In terms of the balance between multiple filter groups—with each filter's TMI score being an indicator of its importance—we would prefer clustering results where all filter groups have a similar TMI score. If one filter group's TMI score is significantly larger than another filter group, the former group will presumably contain more "important filters" than the latter group; and when we proceed to prune a certain ratio of grouped kernels out of both groups, the network will lose more "important kernels" and damage the accuracy retention. In other words, we want to minimize the intervals between the TMI scores of all filter groups. So for a clustering result $\boldsymbol{G}^\ell$, assume $\boldsymbol{g}_j^\ell, ..., \boldsymbol{g}_i^\ell$ are filter groups of $\boldsymbol{G}^\ell$, sorted in descending order according to their TMI scores. We denote the intervals of $\boldsymbol{G}^\ell$ with respect to winning ticket k as:

$$\text{Interval}(\boldsymbol{G}^\ell, k) = [\text{TMI}(\boldsymbol{g}_j^\ell, k) - \text{TMI}(\boldsymbol{g}_{j-1}^\ell, k),\ \ldots,\ \text{TMI}(\boldsymbol{g}_{i+1}^\ell, k) - \text{TMI}(\boldsymbol{g}_i^\ell, k)], \tag{3}$$

where the clustering result $\boldsymbol{G}^\ell = \arg\min_{\boldsymbol{G}^\ell}(\text{Sum}[\text{Interval}(\boldsymbol{G}^\ell, k)])$ is preferred. To further identify the clustering result with the most balanced filter groups in terms of TMI scores, we also want to minimize the variance of intervals between the TMI scores of all filter groups. By combining the two heuristics with a balancing parameter α, the overall scoring system for a clustering result $\boldsymbol{G}^\ell$ is defined as:

$$\underset{\text{filters clustering}}{\text{Score}}(\boldsymbol{G}^\ell, k) = R(\text{Sum}[\text{Interval}(\boldsymbol{G}^\ell, k)]/(n-1)) + \alpha \cdot R(\text{Var}[\text{Interval}(\boldsymbol{G}^\ell, k)]), \tag{4}$$

where the function R denotes the rank of the current $\boldsymbol{G}^\ell$ against other proposed clustering results. For example, if $\text{Sum}[\text{Interval}(\boldsymbol{G}^\ell, k)]$ is the largest among all of the proposed clustering results, then $R(\text{Sum}[\text{Interval}(\boldsymbol{G}^\ell, k)]) = 1$. We choose to use rank instead of the raw value as we are only interested in if one clustering result is better than another on one particular criterion (interval mean or interval variance), not by how much. Therefore by projecting both criteria to rank indices, we avoid the imbalance between the raw values of the two criteria. The $\arg\min$ of this equation yields the best $\boldsymbol{G}^\ell$.

Increase robustness with multiple ticket evaluations Now we defined how to identify the preferred clustering scheme from all proposed $\boldsymbol{G}^\ell$s in layer $\boldsymbol{W}^\ell$, the last challenge is to find the winning ticket k for TMI score calculations. Unfortunately, finding the winning ticket requires an extremely computationally intensive process called *iterative magnitude pruning*, in which multiple training and pruning cycles are involved Frankle et al. (2020). To make matters worse, since we know that there are often multiple winning tickets available from the training process (Renda et al., 2020), picking one ticket but not the other will potentially yield a different set of preferred clustering schemes for the network. It makes the k both a hard-to-find task and a sensitive parameter to tune.

We address both challenges by relying on the same finding from Renda et al. (2020), which demonstrated that for any k where $0 < k_1 \leq k \leq k_2 < t$, $\boldsymbol{W}_k^{'}$ can be a winning ticket, yet this tickets window $[k_1, k_2]$ is shown to be robust for the similar models across different datasets and pruning methods. Thus, we can decide which network model we will prune on, identify its ticket window by consulting experimental results from Renda et al. (2020), and truncate some epochs in such window to conduct multiple evaluations. The optimal clustering scheme for $\boldsymbol{W}^\ell$ will be the one that most often yields the preferred clustering scheme per different k. By doing this, we first avoid searching for winning ticket k, which greatly reduces the computational power needed to implement our method. Secondly, it increases the robustness of our method, since the optimal clustering scheme per layer is no longer sensitive to the choice of k. Last, granted the wide range of $[k_1, k_2]$ observed from most networks, practically we don't even need to search for the exact (k_1, k_2) pair; an approximated guess with a reasonable overlap with the ideal (k_1, k_2) would often suffice[1].

[1]Renda et al. (2020) has no experiment on ResNet-101, so we truncate the tickets windows from ResNet-50 and apply it on ResNet-101 as a form of "guessing." The results are consistent with our other experiments 2.

3.3 EVALUATE CANDIDATE PRESERVED GROUPED KERNEL QUEUES

With filters of $\boldsymbol{W}^\ell$ grouped into filter groups $\{\boldsymbol{g}_1^\ell, \boldsymbol{g}_2^\ell, ..., \boldsymbol{g}_n^\ell\}$, we will carry out the pruning procedure within each filter group. Let $\boldsymbol{K}_i^{\boldsymbol{g}^\ell}$ be the i-th grouped kernel in filter group $\boldsymbol{g}^\ell$ (for $1 \leq i \leq \boldsymbol{C}_{in}^\ell$) such that $\boldsymbol{K}_i^{\boldsymbol{g}^\ell}$ is the collection of the i-th kernels from every filter in $\boldsymbol{g}_1^\ell$. The pruning of a filter group $\boldsymbol{g}^\ell$ can be viewed as the task of identifying a set of indices in group kernels that need to be preserved. To start off such identification, we must first hypothesise a subset of grouped kernels that are most "distinctive" from each other, which helps to preserve the representation power of the original filter group. We define a distance matrix between grouped kernels from $\boldsymbol{g}^\ell$ as $D(\boldsymbol{g}^\ell))$ where $D(\boldsymbol{g}^\ell)_{[u,v]} = |\boldsymbol{K}_u^{\boldsymbol{g}^\ell} - \boldsymbol{K}_v^{\boldsymbol{g}^\ell}|$.

This $D(\boldsymbol{g}^\ell))$ shall be a symmetric matrix with its diagonal full of 0s. With this matrix, we may now translate this problem as a *maximum edge-weight connected subgraph* problem: Given a filter group $\boldsymbol{g}^\ell$ represented as an undirected complete graph $G(V, E)$ with edge weights $w(u, v) = D(\boldsymbol{g}^\ell)_{[u,v]}$ between nodes u and v, find a subset $V^* \subseteq V$ with $|V^*| = k$ where the total edge sum of $G[V^*] := (V^*, E \cap \binom{V^*}{2})$ is maximal.

The brute force solution of this problem requires iterating through every possible subset of grouped kernels with size k and calculate their edge sum. With $\binom{|V|}{k}$ possible subsets available and each subset having $\binom{k}{2}$ edges, this procedure has a time complexity of $\mathcal{O}(k^2 \cdot |V|^k)$. This can be quite a compute-extensive task given each graph $G(V, E)$ deduced from $\boldsymbol{g}^\ell$ has a $|V|$ equals to C_{in}^ℓ, which can be as large as `512` in models like ResNet-101. To address this, we hereby propose a simple greedy algorithm with multiple restarts that may approximate the V^* in question.

Our approximated solution was based on the hypothesis that assumes we already have a set of nodes, $U^* \subset V^*$, representing grouped kernels which will be preserved. To find the next node $u_{\text{next}} \in V^*$ for $u_{\text{next}} \notin U^*$, such u_{next} should have a maximal edge sum to all nodes in U^*. Namely, $u_{\text{next}} = \arg\max_u(\sum_{u \in U^*} w(u, u_{\text{next}}))$. With this, for a filter group $\boldsymbol{g}^\ell$ we may simply set the first grouped kernel $u_{\text{initial}} \in U^*$ to be every grouped kernel $\boldsymbol{K}_i^{\boldsymbol{g}^\ell} \in \boldsymbol{g}^\ell$, find the u_{next} until $|V^*| = k$, and obtain C_{in}^ℓ pruning strategies. This trick is known as *multiple restarts* and was widely adopted in many algorithms, such as k-means clustering.

Algorithm 1 Generate C_{in}^ℓ grouped kernel pruning strategy

Input: Filter group $\boldsymbol{g}^\ell$, PR ▷ one of the n filter group in layer $\boldsymbol{W}^\ell$, pruning ratio
Initialize: Empty list PS ▷ for storing grouped kernels pruning strategies
Initialize: C_{in}^ℓ, $D(\boldsymbol{g}^\ell)$ ▷ number of grouped kernel in and distance matrix of $\boldsymbol{g}^\ell$
1: **for each** $\boldsymbol{K}_i \in \boldsymbol{g}^\ell$ **do**
2: **Initialize:** Empty list U^* ▷ storing preserved grouped kernels per the current strategy
3: **Initialize:** Zero-filled matrix $M \in \mathbb{R}^{(C_{in}^\ell - 1) \times C_{in}^\ell}$ ▷ the cost matrix
4: $M_{[0,j]} \leftarrow w(\boldsymbol{K}_i, \boldsymbol{K}_j)$ for all $\boldsymbol{K_j} \in \boldsymbol{g}^\ell / \boldsymbol{K}_i$
5: U^*.append($\boldsymbol{K}_i$) ▷ make $\boldsymbol{K}_i$ the first grouped kernel to keep
6: **for** `row` $\leftarrow 1$ to C_{in}^ℓ **do**
7: $\boldsymbol{K}_{\text{last}} \leftarrow U^*[-1]$
8: **for each** grouped kernel $\boldsymbol{K} \in \boldsymbol{g}^\ell$ but $\boldsymbol{K} \notin U^*$ **do**
9: `K_index` $\leftarrow \boldsymbol{g}^\ell$.`getindex`($\boldsymbol{K}$)
10: $M_{[\texttt{row},\ \texttt{K_index}]} \leftarrow M_{[\texttt{row}-1,\ \texttt{K_index}]} + w(\boldsymbol{K}_{\text{last}}, \boldsymbol{K})$
11: U^*.`append`($\arg\max_{\boldsymbol{K}}(M_{[\texttt{row}]})$)
12: **if** $|U^*| == (1 - PR)C_{in}^\ell$ **then**
13: $V^* \leftarrow U^*$
14: PS.`append`(V^*) and **continue**
15: **return** PS

The core of our algorithm resides at line `10`, where we use the computed result of previous row in M to reduce time complexity of our algorithm. With $M \in \mathbb{R}^{(C_{in}^\ell - 1) \times C_{in}^\ell}$ and filled C_{in}^ℓ times, our approximation algorithm therefore has a time complexity of $\mathcal{O}(|V|^3)$ (as $|V| = C_{in}^\ell$). This provides

a sensible improvement upon the $\mathcal{O}(k^2 \cdot |V|^k)$ brute force solution. To further and better identify the best grouped kernel pruning strategy among the C_{in}^{ℓ} candidates strategy in PS, we define the score of a strategy V^* on $\boldsymbol{g}^\ell$ as the following:

$$\underset{\text{grouped kernel pruning}}{\text{Score}}(V^*, \boldsymbol{g}^\ell) = \sum_{k_u, k_v \in \binom{V^*}{2}} w(k_u, k_v) - \beta \Big(\sum_{p \in \boldsymbol{g}^\ell / V^*} \big(\sum_{k_{i=1} \in V^*}^{k_\gamma} w(p, k_i) \big) \Big). \tag{5}$$

The first term calculates the inner distance sum of V^*, where a larger value represents greater inner heterogeneity within the kept grouped kernels. The second term iterates through every pruned grouped kernel p, finds the γ number of kept grouped kernels that are closest to each p according to the distance matrix $D(\boldsymbol{g}^\ell)$, and sums over the distance between each p to its corresponding γ kept grouped kernels. Where a pruning strategy with a smaller second term has better homogeneity between kept and pruned grouped kernels. Since a strategy with greater inner heterogeneity and smaller outer homogeneity is desired, we further introduce a tunable parameter $-\beta$ to balance two terms and define the best the pruning strategy of $\boldsymbol{g}^\ell$ to be $V^*_{\text{best}} = \underset{V^*}{\arg\max}(\text{Score}(V^*, \boldsymbol{g}^\ell))$ for all V^* available in PS.

3.4 Layer reconstruction to grouped convolution architecture for fine-tuning

The layer reconstruction of our method happens at two places: 1) Given a convolutional layer $\boldsymbol{W}^\ell$, we permute the output channels of such layer according to the clustering result of filters groups. This makes filters within $\boldsymbol{W}^\ell$ form n grouped filters as demonstrated in Figure 1 (b). 2) After a certain ratio of grouped kernels were pruned from their corresponding filter groups, we permute the input channels of the remaining grouped kernels to remove sparsity from each filter group (Figure 1(c) to (d)).

As the result of the above two channel permutations, we will have n dense grouped filters left in the pruned $\boldsymbol{W}^\ell$. Since every grouped filter has its own set of input-output pathways, the pruned $\boldsymbol{W}^\ell$ is essentially a grouped convolution architecture. This architecture enables parallel computing capabilities, because every grouped filter can be deployed to a different device so that all group filters may compute simultaneously.

Please refer to the Appendix for the general procedure of our method in pseudocode.

4 Experiments and Results

4.1 Experiment Settings

Network Architectures and Datasets We evaluate the efficacy of our method on popular networks with various depths and architectures: ResNet-20/32/56/110 with the `BasicBlock` implementation and ResNet-110 with the `BottleNeck` implementation. For datasets, we choose CIFAR-10 and Tiny-ImageNet as they vary in complexity, and Wang et al. (2021a) has carried out a fairly large-scale comparative experiment of many different structured pruning algorithms running under the same settings—which provides us with a rich background to compare against.

Compared Methods and Performance Evaluation We test our method against various existing pruning methods showed in Table 2. All mentioned pruning methods are evaluated against the following criteria: ΔAcc, $\downarrow$ Params, and $\downarrow$ FLOPs which are defined as the difference of top-1 accuracy, parameters reduction, and FLOPs reduction between the baseline and pruned model. In addition, we investigate the pruning procedure and pruning setting of the compared methods in the three following aspects: 1) Does the method require an iterative pruning procedure? 2) Does the method require a special fine-tuning setup that deviated from 3.2.2? 3) What's the fine-tuning budget of the method-in-question? We believe these are crucial information for building a fair understanding of the three aforementioned numerical criteria.

Training and Pruning Settings For all networks and datasets, we train the baseline models for 300 epochs with the learning rate starts at 0.1 and divide by 10 per every 100 epochs. The model was trained using SGD with a weight-decay set to $5e-4$, momentum set to 0.9, and a batch-size of 64. Our pruning settings are largely identical to our training settings except for the learning rate, which is set to 0.01 at the start. For experiments on Tiny-ImageNet, an extra 50 epochs of fine-tuning budget is granted (for a total of 350 epochs spent on fine-tuning).

4.2 Results and analysis

As demonstrated in Table 2, **our method yields superior accuracy retention than various modern SOTA methods across all six tested network-dataset combinations with a commensurate amount of parameters pruned**[2]. Specifically, our method achieves such results with a one-shot pruning procedure and a standard fine-tuning setup. Which are usually considered two less advantageous design as recent SOTA methods Wang et al. (2021a); Frankle & Carbin (2019) utilize iterative pruning, yet some simple tricks like *weight reinitialization* before fine-tuning may often provide most algorithms another performance boost (Renda et al., 2020). We opt not to implement such tricks to provide a cleaner delivery of our method. Last, although not universally observed, our method often requires a smaller fine-tuning budget than other offerings. **Here we include an abbreviated experiment report, please refer to Table 2 for the full one.**

Table 1: Abbreviated results with ResNet-56/110 on CIFAR-10. "IT", "SF" respectively indicate if the method-in-question requires an iterative pruning or a special fine-tuning procedure. "FB" is the fine-tuning budget of each method (in terms of # of epochs). "BA" represents the pre-pruned network's accuracy. A cell with "-" implies either such information is inapplicable or we failed to confidently identify such information. Methods noted with $\odot$ are replicated by Wang et al. (2021a), the rests are drawn from their original papers.

	IT	FB	SF	BA (%)	Δ Acc (%)	$\downarrow$ Params (%)	$\downarrow$ FLOPs(%)
ResNet-56 on CIFAR-10: FLOPs: 1.27E8 Params: 8.53E5							
DBP$^{\odot}$ (Wang et al., 2019b)	✓	560	✓	93.69	$\downarrow$0.42	40	52
FPGM (He et al., 2019)	✓	200	✓	93.59	$\downarrow$0.33	-	52
GAL (Lin et al., 2019b)	✗	-	✗	93.26	$\uparrow$0.12	12	38
PScratch (Lym et al., 2019)	-	-	✓	93.23	$\downarrow$0.18	-	50
SFP (He et al., 2018)	✓	-	✓	93.59	$\downarrow$1.33	-	53
3D (Wang et al., 2021b)	✓	560	✓	93.69	$\uparrow$0.07	40	50
TMI-GKP (ours)	✗	300	✗	93.78	$\uparrow$**0.16**	43.49	43.23
ResNet-110 on CIFAR-10: FLOPs: 2.55E8 Params: 1.73E6							
DHP (Li et al., 2020)	-	-	✓	94.69	$\downarrow$0.06	37	36
FPGM (He et al., 2019)	✓	200	✓	93.68	$\uparrow$0.05	-	52
GAL (Lin et al., 2019b)	✗	-	✗	93.50	$\downarrow$0.76	45	49
PScratch (Lym et al., 2019)	-	-	✓	93.49	$\uparrow$0.20	-	40
SFP (He et al., 2018)	✓	-	✓	93.68	$\downarrow$0.30	-	41
TMI-GKP (ours)	✗	300	✗	94.26	$\uparrow$**0.64**	43.52	43.31

4.3 Ablation study

We anatomize the effectiveness of different components of our method and compare their performance against some common alternatives. Please refer to Appendix: Ablation Study Experiments for details.

5 Conclusion

We proposed a pruning framework capable of yielding SOTA performance with a straightforward procedure and often less costly. Though the ablation study has demonstrated the positive contribution of our method at a component-level, we believe the power of densely structured kernel pruning may go well beyond our method; yet there will certainly be more—and hopefully better—inductive bias and unsupervised clustering schemes made available. The framework we proposed may thrive on these further-discoveries and therefore have a longer life-span.

[2]For viewers' attention, we do notice some irregularities regarding the experiment combination of ResNet-101 and Tiny-ImageNet, which we include a discussion in the Appendix A.2.

REFERENCES

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 254–263. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/arora18b.html.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*. OpenReview.net, 2019. URL http://dblp.uni-trier.de/db/conf/iclr/iclr2019.html#FrankleC19.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3259–3269. PMLR, 2020. URL http://proceedings.mlr.press/v119/frankle20a.html.

Qingbei Guo, X. Wu, J. Kittler, and Zhi quan Feng. Self-grouping convolutional neural networks. *Neural networks : the official journal of the International Neural Network Society*, 132:491–505, 2020.

Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing XU, and Tong Zhang. Model rubik's cube: Twisting resolution, depth and width for tinynets. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 19353–19364. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/e069ea4c9c233d36ff9c7f329bc08ff1-Paper.pdf.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In Jérôme Lang (ed.), *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pp. 2234–2240. ijcai.org, 2018. doi: 10.24963/ijcai.2018/309. URL https://doi.org/10.24963/ijcai.2018/309.

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 4340–4349. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00447. URL http://openaccess.thecvf.com/content_CVPR_2019/html/He_Filter_Pruning_via_Geometric_Median_for_Deep_Convolutional_Neural_Networks_CVPR_2019_paper.html.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 1398–1406. IEEE Computer Society, 2017. doi: 10.1109/ICCV.2017.155. URL https://doi.org/10.1109/ICCV.2017.155.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

Forrest N. Iandola, Matthew W. Moskewicz, K. Ashraf, Song Han, W. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡1mb model size. *ArXiv*, abs/1602.07360, 2017.

Elsie Maria Jordaan. Development of robust inferential sensors: Industrial application of support vector machines for regression. 2004.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rJqFGTslg.

Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. DHP: differentiable meta pruning via hypernetworks. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part VIII*, volume 12353 of *Lecture Notes in Computer Science*, pp. 608–624. Springer, 2020. doi: 10.1007/978-3-030-58598-3_36. URL https://doi.org/10.1007/978-3-030-58598-3_36.

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pp. 1526–1535. Computer Vision Foundation / IEEE, 2020. doi: 10.1109/CVPR42600.2020.00160. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Lin_HRank_Filter_Pruning_Using_High-Rank_Feature_Map_CVPR_2020_paper.html.

Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019a.

Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David S. Doermann. Towards optimal structured CNN pruning via generative adversarial learning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 2790–2799. Computer Vision Foundation / IEEE, 2019b. doi: 10.1109/CVPR.2019.00290. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Lin_Towards_Optimal_Structured_CNN_Pruning_via_Generative_Adversarial_Learning_CVPR_2019_paper.html.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019a. URL https://openreview.net/forum?id=rJlnB3C5Ym.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=rJlnB3C5Ym.

Sangkug Lym, Esha Choukse, Siavash Zangeneh, Wei Wen, Mattan Erez, and Sujay Shanghavi. Prunetrain: Gradual structured pruning from scratch for faster neural network training. *CoRR*, abs/1901.09290, 2019. URL http://arxiv.org/abs/1901.09290.

Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. PCONV: the missing but desirable sparsity in DNN weight pruning for real-time execution on mobile devices. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 5117–5124. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/5954.

Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J. Dally. Exploring the granularity of sparsity in convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJGCiw5gl.

Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. Touretzky (ed.), *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1989. URL https://proceedings.neurips.cc/paper/1988/file/07e1cd7dca89a1678042477183b7ac3f-Paper.pdf.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1gSj0NKvB.

Erich Schubert and Arthur Zimek. ELKI: A large open-source library for data analysis - ELKI release 0.7.5 "heidelberg". *CoRR*, abs/1902.03616, 2019. URL http://arxiv.org/abs/1902.03616.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6377–6389. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/46a4378f835dc8040c8057beb6a2da52-Paper.pdf.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, 2018:1–13, 2018.

Wenxiao Wang, Cong Fu, Jishun Guo, Deng Cai, and Xiaofei He. COP: customized deep model compression via regularized correlation-based filter-level pruning. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 3785–3791. ijcai.org, 2019a. doi: 10.24963/ijcai.2019/525. URL https://doi.org/10.24963/ijcai.2019/525.

Wenxiao Wang, Shuai Zhao, Minghao Chen, Jinming Hu, Deng Cai, and Haifeng Liu. DBP: discrimination based block-level pruning for deep model acceleration. *CoRR*, abs/1912.10178, 2019b. URL http://arxiv.org/abs/1912.10178.

Wenxiao Wang, Shuai Zhao, Minghao Chen, Jinming Hu, Deng Cai, and Haifeng Liu. Dbp: Discrimination based block-level pruning for deep model acceleration. *ArXiv*, abs/1912.10178, 2019c.

Wenxiao Wang, Minghao Chen, Shuai Zhao, Long Chen, Jinming Hu, Hai Liu, Deng Cai, Xiaofei He, and Wei Liu. Accelerate cnns from three dimensions: A comprehensive pruning framework. In *ICML 2021*, July 2021a.

Wenxiao Wang, Minghao Chen, Shuai Zhao, Long Chen, Jinming Hu, Haifeng Liu, Deng Cai, Xiaofei He, and Wei Liu. Accelerate cnns from three dimensions: A comprehensive pruning framework. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10717–10726. PMLR, 2021b. URL http://proceedings.mlr.press/v139/wang21e.html.

Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin. Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5359–5368. PMLR, 2018. URL http://proceedings.mlr.press/v80/wu18h.html.

Niange Yu, Shi Qiu, Xiaolin Hu, and Jianmin Li. Accelerating convolutional neural networks by group-wise 2d-filter pruning. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2502–2509, 2017. doi: 10.1109/IJCNN.2017.7966160.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. NISP: pruning networks using neuron importance score propagation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 9194–9203. Computer Vision Foundation / IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00958. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Yu_NISP_Pruning_Networks_CVPR_2018_paper.html.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 3592–3602, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/1113d7a76ffceca1bb350bfe145467c6-Abstract.html.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017.

A Appendix

A.1 Supplementary material on Proposed Method

A.1.1 *Silhouette score* and Accuracy Retention

We experiment ResNet-32 on CIFAR-10 (BA $= 92.82\%$) with K-PCA + Equal-sized k-Means and Spectral + Equal-sized k-Means. The mean *Silhouette score* of the latter combination is $0.3349/0.0268 = 12.49$ times higher than the previous combination, indicating that—if the Silhouette score may capture the better clustering result with respect to accuracy retention—the latter approach should be significantly better. However, the K-PCA approach has a pruned accuracy of 92.68% where the Spectral approach has a pruned accuracy of 92.41%, a $+0.27\%$ in favor of the method which the Silhouette score strongly suggests against. This demonstrated that tools like the Silhouette score might not be a good judge of clustering quality in terms of accuracy retention.

A.1.2 Effectiveness of our greedy approximation algorithm

To test out whether our greedy approximation algorithm may yield a solution that is close enough to the optimal one, we conducted a trial of TMI-GKP with $\beta = 0$ on ResNet-32. For the first ten convolutional layers (80 filter groups pruned), all approximated solutions are within 0.5% in terms of the cost difference to the optimal solution—which implies the effectiveness of our approximation algorithm.

A.1.3 General Procedure

A.2 Experiments on Tiny-ImageNet

For transparency, we want to highlight that we noticed our baseline is significantly lower (-4.75%) than other experiments conducted on the combination of ResNet-101 and Tiny-ImageNet. Yet—unlike in other experiments which are not on Tiny-ImageNet—we cannot achieve a comparable FLOPs reduction with a similar or greater amount of parameters pruned (even on the same network). We investigated the training procedure of (Wang et al., 2021a) (as they conducted all compared experiments on this combination) and found out that their baseline was trained with 200 fewer epochs, with one more data augmentation used, while keeping all other settings similar; based on our experience, these kind of setting variations are unlikely to cause this much of a baseline drop. In terms of FLOPs reduction, we suspect we might be using a different counting criteria[3] than the authors of (Wang et al., 2021a), and we alert our readers to be careful on reaching conclusions despite our method delivers significantly better Δ Acc than the second best offering ($+1.09\%$) on ResNet-101 and Tiny-ImageNet.

[3]We utilized the `thop.profile()` method to calculate parameters and FLOPs reduction.

Algorithm 2 General Procedure of TMI-GKP

Input: $\boldsymbol{W}, (k_1, k_2)$ ▷ trained model, tickets window
Input: CS, PR, n ▷ clustering schemes, pruning ratio, number of groups
for each convolutional layer $\boldsymbol{W}^\ell \in \boldsymbol{W}$ **do**
  Initialize: Empty list $\boldsymbol{G}^\ell_{\text{candidates}}$ ▷ To store candidate clustering results
  for $k_i \in [k_1, k_2]$ **do**
    Cluster filters from $\boldsymbol{W}^\ell$ into n equal-sized groups according to each scheme in CS
    Identify the best clustering result $\boldsymbol{G}^\ell_{k_i} \leftarrow \arg\min_{\boldsymbol{G}^\ell}(\underset{\text{filters clustering}}{\text{Score}}(\boldsymbol{G}^\ell, k_i))$ [Equation 4]
    $\boldsymbol{G}^\ell_{\text{candidates}}.\texttt{append}(\boldsymbol{G}^\ell_{k_i})$
  Identify $\boldsymbol{G}^\ell_{\text{best}} \leftarrow$ the most occurred item in $\boldsymbol{G}^\ell_{\text{candidates}}$
  Apply $\boldsymbol{G}^\ell_{\text{best}}$ on $\boldsymbol{W}^\ell$
  for each filter group $\boldsymbol{g}^\ell_i \in \boldsymbol{G}^\ell_{\text{best}}$ **do**
    Get grouped kernel pruning strategies $V^* \leftarrow$ **Algorithm 1**$(\boldsymbol{g}^\ell_i)$ [Algorithm 3.3]
    Identify $V^*_{\text{best}} \leftarrow \arg\max_{V^*}(\underset{\text{grouped kernel pruning}}{\text{Score}}(V^*, \boldsymbol{g}^\ell_i))$ [Equation 5]
    Apply V^*_{best} on $\boldsymbol{g}^\ell_i$
  $\boldsymbol{W}^\ell \leftarrow$ pruned $\boldsymbol{W}^\ell$ reconstructed in grouped convolution architecture
return $\boldsymbol{W}$

A.3 Ablation Study Experiments

For disambiguation, we refer to the filter clustering procedure in TMI-GKP as TMI-driven clustering, and the greedy grouped kernel pruning procedure is noted as greedy-induced pruning).

A.3.1 TMI-driven clustering v. Other filter clustering schemes

We set the greedy grouped kernel pruning procedure to fix and feed our method different schemes on filter clustering: three clustering schemes individually and a uniform random assignment of the three clustering schemes. The results with ResNet-32 on CIFAR-10 suggest the TMI-driven clustering may deliver the best accuracy retention. However, by conducting experiments for the following ablation study, we notice that the combination of ResNet-110 on CIFAR-10, forcing the clustering scheme as *spectral clustering* may achieve slightly superior accuracy retention to TMI-GKP ($+0.06\%$). But by running the same clustering scheme on other experiment combinations like ResNet-20 and ResNet-32 on CIFAR-10, spectral clustering induces much lower results (-0.28% and -0.30%). This implies the TMI-driven clustering may not guarantee finding the best clustering in terms of accuracy retention, but rather a robust policy with more comprehensive considerations done to deliver a "better" solution than many other structured pruning methods when combined with the greedy-induce pruning scheme.

Method	BA (%)	Δ Acc (%)	↓ Params (%)	↓ FLOPs(%)
ResNet-32 on CIFAR-10: FLOPs: 6.95E7 Params: 4.29E7				
TMI Clustering w/ Greedy	92.82	↑ **0.14**	43.43	43.09
K-PCA + k-Means w/ Greedy	92.82	↑ 0.01	43.43	43.09
Spectral + k-Means w/ Greedy	92.82	↓ 0.16	43.43	43.09
t-SNE + k-Means DBSCAN w/ Greedy	92.82	↓ 0.14	43.43	43.09
Random Clustering Schemes w/ Greedy	92.82	↑ 0.03	43.43	43.09

A.3.2 Greedy-induced pruning v. Other grouped kernel pruning schemes

We set the filter clustering scheme to fix spectral clustering and feed our method with different commonly applied grouped kernel pruning schemes (L2 and center pruning). Experiments of ResNet-32/110 on CIFAR-10 have confirmed that our greedy-induced pruning scheme may yield better accuracy retention in a controlled setting.

Method	BA (%)	Δ Acc (%)	↓ Params (%)	↓ FLOPs(%)
ResNet-32 on CIFAR-10: FLOPs: 6.95E7 Params: 4.29E7				
Greedy-induced Pruning	92.82	↑ **0.16**	43.4	43.1
L^2 Pruning	92.82	↑ 0.30	43.4	43.1
Center Pruning	92.82	↓ 0.19	43.4	43.1
ResNet-110 on CIFAR-10: FLOPs: 2.55E8 Params: 1.73E6				
Greedy-induced Pruning	94.26	↑ **0.7**	43.3	43.5
L^2 Pruning	93.76	↓ 0.08	43.3	43.5
Center Pruning	93.76	↓ 0.05	43.3	43.5

A.3.3 TMI-GKP vs. Grouped convolution-only

Since the architecture of grouped convolution itself may induce a parameters to drop, we are also interested in learning how much does it contribute to accuracy retention. We modified the original network as grouped convolution, trained it with the same training settings, and compared it against TMI-GKP. With a comparable amount of parameters and FLOPs reduction, the results achieved with only grouped convolution are well below our TMI-GKP.

Method	BA (%)	Pruned Acc (%)	↓ Params (%)	↓ FLOPs(%)
ResNet-20 on CIFAR-10: FLOPs: 4.09E7 Params: 2.70E5				
Grouped Conv ($n = 2$)	90.48	-	49.0	49.5
TMI-GKP	92.35	**91.96** (↑ 0.39)	43.4	42.9
ResNet-32 on CIFAR-10: FLOPs: 6.95E7 Params: 4.64E5				
Grouped Conv ($n = 2$)	91.75	-	49.2	49.6
TMI-GKP	92.82	**92.96** (↑ 0.14)	43.4	43.1
ResNet-56 on CIFAR-10: FLOPs: 1.27E8 Params: 8.53E5				
Grouped Conv ($n = 2$)	92.34	-	49.4	49.7
TMI-GKP	93.78	**93.94** (↑ 0.16)	43.5	43.2
ResNet-110 on CIFAR-10: FLOPs: 2.55E8 Params: 1.73E6				
Grouped Conv ($n = 2$)	92.9	-	49.5	49.7
TMI-GKP	94.26	**94.90** (↑ 0.64)	43.5	43.3

A.4 Dimensionality Reduction and Clustering Schemes

A.4.1 Kernel Principal Component Analysis (K-PCA)

Kernel PCA is considered an improved version of PCA. The choice of kernel functions will significantly affect the experimental results. Global kernels (e.g. polynomial, Sigmoid) and local kernels (e.g. RBF, Laplacian) are the most commonly used kernel functions for K-PCA. They capture different characteristics of the data: the former one demonstrates better extrapolation abilities, and the latter one has better interpolation abilities (Jordaan, 2004) Since the filters are usually not linearly separable and usually there are fewer filters than the number of features, we will mix the global and local kernel functions to benefit both extrapolation and interpolation properties. Specifically, we mixed a polynomial and an RBF kernel. We use a parameter λ to control the weights of these two functions.

$$\begin{aligned} K_{\text{new}}(x, y) &= \lambda K_{\text{Poly}}(x, y + (1-\lambda) K_{\text{RBF}}(x, y) \\ &= \lambda (a x^{\mathrm{T}} \cdot y + c)^d + (1-\lambda) \exp\left(-\frac{\| x - y \|^2}{2\sigma^2}\right), \end{aligned} \tag{6}$$

where $\lambda \in [0,\ 1)$, parameters a and c in polynomial kernel has been fixed to 1 and 0, $d = 2$ is the degree of polynomial kernel, σ in RBF kernel is the reciprocal of the width of the radial basis function.

A.4.2 Spectral Clustering

Spectral Clustering is yet another maturely studied and widely adopted clustering method. Unlike in K-PCA, for this method we simply utilize the implementation of Von Luxburg (2007). Specifically

in TMI-GKP, we utilize the variation of *mutual-KNN* as the similarity graph and cosine distance as the adjacency matrix based on empirical observations.

A.4.3 EQUAL-SIZED k-MEANS

To ensure we may have n equal-sized groups out of each filter group, we utilize the default *Same-size k-Means Variation* offered by Schubert & Zimek (2019).

A.4.4 T-SNE AND DBSCAN

We explore the option of t-SNE and DBSCAN because we want to add a density-based clustering scheme from the two distance-based offerings. For implementation details: we set the `perplexity` of t-SNE to be the size of the filter group (namely, C_{in}^{ℓ}/n in layer $\boldsymbol{W}^{\ell}$) and iterate through different combination of ϵ and `min_samples` until the DBSCAN algorithm may yield a clustering result of $\geq n+1$ groups. After that, we execute the same data-point reassignment procedure as listed in Schubert & Zimek (2019) until n equal-sized groups are achieved.

Table 2: Full comparisons of different pruning methods with ResNet-20/32/56/110 on CIFAR-10 and ResNet-56/101 on Tiny-ImageNet. "IT", "SF" respectively indicate if the method-in-question requires an iterative pruning or a special fine-tuning procedure. "FB" is the fine-tuning budget of each method (in terms of # of epochs). "BA" represents the pre-pruned network's accuracy. A cell with "-" implies either such information is inapplicable or we failed to confidently identify such information. Methods noted with $^{\odot}$ are replicated by Wang et al. (2021a), the rests are drawn from their original papers.

	IT	FB	SF	BA (%)	Δ Acc (%)	$\downarrow$ Params (%)	$\downarrow$ FLOPs(%)
ResNet-20 on CIFAR-10: FLOPs: 4.09E7 Params: 2.70E5							
DHP (Li et al., 2020)	-	-	✓	92.54	$\downarrow$ 1.00	44	48
FPGM (He et al., 2019)	✓	200	✓	92.20	$\downarrow$ 1.11	-	42
PScratch (Lym et al., 2019)	-	-	✓	91.75	$\downarrow$ 1.20	-	50
Rethink (Liu et al., 2019a)	✓	40	✓	92.41	$\downarrow$ 1.34	-	40
SFP (He et al., 2018)	✓	-	✓	92.20	$\downarrow$ 1.37	-	42
TMI-GKP (ours)	✗	300	✗	92.35	$\downarrow$ **0.39**	43.35	42.87
ResNet-32 on CIFAR-10: FLOPs: 6.95E7 Params: 4.64E5							
DBP$^{\odot}$ (Wang et al., 2019b)	✓	560	✓	93.18	$\downarrow$ 0.53	28	48
FPGM (He et al., 2019)	✓	200	✓	92.63	$\downarrow$ 0.32	-	42
GAL$^{\odot}$ (Lin et al., 2019b)	✗	560	✗	93.18	$\downarrow$ 1.46	39	50
PScratch$^{\odot}$ (Lym et al., 2019)	-	-	✓	93.18	$\downarrow$ 1.00	-	50
SFP (He et al., 2018)	✓	-	✓	92.63	$\downarrow$ 0.55	-	42
3D (Wang et al., 2021b)	✓	560	✓	93.18	$\uparrow$ 0.09	38	49
TMI-GKP (ours)	✗	300	✗	92.82	$\uparrow$ **0.14**	43.43	43.09
ResNet-56 on CIFAR-10: FLOPs: 1.27E8 Params: 8.53E5							
CP (He et al., 2017)	-	20	✗	92.80	$\downarrow$ 1.90	-	50
DBP$^{\odot}$ (Wang et al., 2019b)	✓	560	✓	93.69	$\downarrow$ 0.42	40	52
DHP$^{\odot}$ (Li et al., 2020)	-	-	✓	93.65	$\downarrow$ 0.07	42	49
FPGM (He et al., 2019)	✓	200	✓	93.59	$\downarrow$ 0.33	-	52
GAL (Lin et al., 2019b)	✗	-	✗	93.26	$\uparrow$ 0.12	12	38
HRank (Lin et al., 2020)	-	30 per layer	-	93.26	$\downarrow$ 0.09	42	50
PScratch (Lym et al., 2019)	-	-	✓	93.23	$\downarrow$ 0.18	-	50
Rethink (Liu et al., 2019a)	✓	40	✓	93.80	$\downarrow$ 0.73	-	50
SFP (He et al., 2018)	✓	-	✓	93.59	$\downarrow$ 1.33	-	53
3D (Wang et al., 2021b)	✓	560	✓	93.69	$\uparrow$ 0.07	40	50
TMI-GKP (ours)	✗	300	✗	93.78	$\uparrow$ **0.16**	43.49	43.23
ResNet-110 on CIFAR-10: FLOPs: 2.55E8 Params: 1.73E6							
CP (He et al., 2017)	-	20	✗	92.80	$\downarrow$ 1.90	-	50
DHP (Li et al., 2020)	-	-	✓	94.69	$\downarrow$ 0.06	37	36
FPGM (He et al., 2019)	✓	200	✓	93.68	$\uparrow$ 0.05	-	52
GAL (Lin et al., 2019b)	✗	-	✗	93.50	$\downarrow$ 0.76	45	49
PFEC (Li et al., 2017)	✗	-	✓	93.53	$\downarrow$ 0.61	-	39
PScratch (Lym et al., 2019)	-	-	✓	93.49	$\uparrow$ 0.20	-	40
Rethink (Liu et al., 2019a)	✓	40	✓	93.77	$\uparrow$ 0.15	-	40
SFP (He et al., 2018)	✓	-	✓	93.68	$\downarrow$ 0.30	-	41
TMI-GKP (ours)	✗	300	✗	94.26	$\uparrow$ **0.64**	43.52	43.31
ResNet-56 on Tiny-ImageNet: FLOPs: 5.06E8 Params: 8.65E5							
DBP$^{\odot}$ (Wang et al., 2019b)	✓	420	✓	56.55	$\downarrow$ 0.98	25	53
DHP$^{\odot}$ (Li et al., 2020)	-	420	✓	56.55	$\downarrow$ 0.73	46	55
GAL$^{\odot}$ (Lin et al., 2019b)	✗	420	✗	56.55	$\downarrow$ 0.68	32	52
3D (Wang et al., 2021b)	✓	420	✓	56.55	$\downarrow$ 0.51	34	59
TMI-GKP (ours)	✗	350	✗	52.68	$\downarrow$ **0.20**	36.74	37.05
ResNet-101 on Tiny-ImageNet: FLOPs: 1.08E10 Params: 4.29E7							
DBP$^{\odot}$ (Wang et al., 2019b)	✓	420	✓	64.83	$\downarrow$ 3.48	76	77
DHP$^{\odot}$ (Li et al., 2020)	-	420	✓	64.83	$\downarrow$ 0.01	50	75
GAL$^{\odot}$ (Lin et al., 2019b)	✗	420	✗	64.83	$\downarrow$ 0.50	45	76
3D (Wang et al., 2021b)	✓	420	✓	64.83	$\uparrow$ 0.44	51	75
TMI-GKP (ours)	✗	350	✗	60.08	$\uparrow$ **1.53**	43.53	43.25