One-Cycle Structured Pruning via Stability-Driven Subnetwork Search

Deepak Ghimire[1] **Dayoung Kil**[2] **Seonghwan Jeong**[1] **Jaesik Park**[3] **Seong-heum Kim**[2*]
[1]IT Application Research Center, Korea Electronics Technology Institute
[2]Department of Intelligent Semiconductors, Soongsil University
[3]Department of Computer Science & Engineering, Seoul National University
{deepak, shjeong}@keti.re.kr, dayoung-kil@naver.com, seongheum@ssu.ac.kr, jaesik.park@snu.ac.kr

Abstract

Existing structured pruning typically involves multi-stage training procedures that often demand heavy computation. Pruning at initialization, which aims to address this limitation, reduces training costs but struggles with performance. To address these challenges, we propose an efficient one-cycle structured pruning framework that integrates pre-training, pruning, and fine-tuning into a single training cycle without compromising model performance, referred to as the 'one-cycle approach'. The core idea is to search for the optimal sub-network during the early stages of network training, guided by norm-based group saliency criteria and structured sparsity regularization. We introduce a novel pruning indicator that identifies the stable pruning epoch by measuring the similarity between evolving pruning sub-networks across consecutive training epochs. Additionally, group sparsity regularization helps accelerate the pruning process, thereby speeding up overall training. Extensive experiments on the CIFAR-10/100 and ImageNet datasets using VGGNet, ResNet, and MobileNet architectures demonstrate that our method achieves state-of-the-art accuracy while being one of the most efficient pruning frameworks in terms of training time. Our code is available at https://github.com/ghimiredhikura/OCSPruner.

1. Introduction

Over the past decade, deep neural networks (DNNs) have demonstrated superior performance at the expense of substantial memory usage, computing power, and energy consumption [8, 18, 22, 63]. For compressing and deploying these complex models on low-capability devices, pruning has emerged as a particularly promising approach for many embedded and general-purpose computing platforms. Structured pruning [5, 6, 14, 33, 51, 55] and unstructured pruning [11, 15, 20, 50] represent the two predominant methods in pruning. Also, there has been increasing interest in middle-level pruning granularity [39, 46, 65], which provides the capability to achieve fine-grained structured sparsity, combining the advantages of both unstructured fine-grained sparsity and structured coarse-grained sparsity.

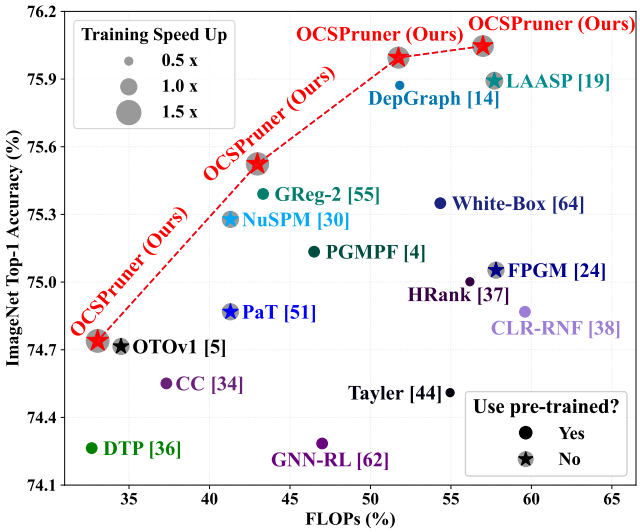

Figure 1. Pruning ResNet50 on the ImageNet dataset. Each method is shown as a colored circle, with an overlaid star indicating the variant trained from scratch. The marker size reflects the training speedup relative to the baseline training cost.

With the emergence of large-scale DNN models [3, 45, 63], there is a growing need to reassess the conventional pruning approach of pre-training, pruning, and fine-tuning, which demands excessive resources. Conventional pruning often requires multi-stage pipelines with repeated fine-tuning procedures or offline re-training. Without efficient learning mechanisms, it is not only a waste of computational power and time but also less applicable to real-time scenarios. One-cycle training and pruning, also known as one-cycle pruning, presents itself as a viable alternative solution to this problem. It is therefore important to explore whether a one-cycle pruning framework can quickly adapt

*Corresponding author.

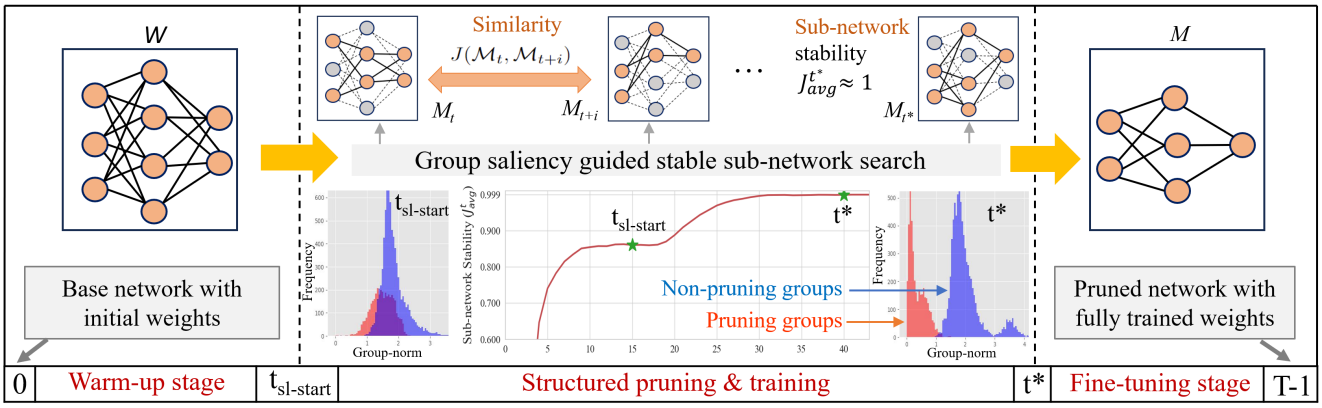

Figure 2. Overview of the OCSPruner algorithm. The process begins with training the baseline model from scratch. As training progresses, from $t_{\text{sl-start}}$ epoch, the pruning sub-structure gradually stabilizes as indicated by the sub-network stability score computed over consecutive training epochs. Final pruning occurs at the stable epoch t^*, followed by the remaining training epochs to converge the pruned structure.

to new tasks or environments in real-world applications such as autonomous vehicles, drones, and robots [5, 16].

To address the limitations of conventional pruning, we introduce **O**ne-**C**ycle **S**tructured **Pruner** (**OCSPruner**), which eliminates the requirement for pre-trained base network for pruning. Considering scenarios where networks need to learn from scratch within limited resource budgets, the efficiency gained from avoiding pre-training the base model becomes more compelling. Hence, we focus on efficient and stable structured pruning, ensuring model performance in a more sustainable way. Basically, structured pruning offers the advantage of creating slimmer yet structured networks that do not rely on specific hardware or software; however, it can occasionally lead to performance drops, particularly when dealing with high pruning ratios.

Considering the insights from the Lottery Ticket Hypothesis [15], which suggests that sparse, trainable subnets can be identified within larger networks, the timing and method of pruning become critical. Our focus on one-cycle structured pruning involves pruning the network at initialization [7, 15, 31]. However, this type of method may suffer from potential sub-optimal network structures that result in performance degradation. Thus, rather than pruning at initialization, our proposed strategy involves early-phase pruning during training while still maintaining a one-cycle approach. Despite related methods like pruning-aware training (PaT) [51] and loss-aware automatic selection of pruning criteria (LAASP) [19], they overlook structural parameter grouping during saliency estimation, leading to reduced performance [14]. Specifically, PaT [51] focuses solely on total channel numbers, not considering individual channel identities during stable sub-network selection, while LAASP [19] lacks stability estimation criteria and involves complex iterative pruning. This paper addresses these limitations and introduces additional algorithmic enhancements for early-phase structured pruning during training from scratch, resulting in state-of-the-art performance.

Fig. 2 illustrates the overall flow of the proposed pruning algorithm. Our algorithm begins with network training from scratch over a few epochs. Following this, we applied structured sparsity regularization targeting pruning groups of parameters alongside network training. For each training epoch, we proposed a novel method to measure the stability of network pruning, leveraging the similarity among temporarily pruned sub-structures across consecutive training epochs. The sub-structures are determined by globally partitioning the structurally grouped parameters into pruning and non-pruning groups utilizing norm-based group saliency scores. As training progresses, the penalty factor for regularization progressively increases, thereby promoting the partitioned group of pruning parameters to approach zero gradually. Finally, the proposed algorithm automatically determines the epoch of stable pruning, during which final pruning is executed. This will occur when the partitioning of parameters into pruning and non-pruning groups achieves stability across successive training epochs. Following this, standard training will persist with the pruned sub-structure for the remaining epochs. This approach leads to an efficient one-cycle training framework, achieving a fully trained pruned network starting from the randomly initialized base network.

The main contributions are summarized as follows:

- We proposed an efficient structured pruning algorithm that combines traditional training, pruning, and fine-tuning steps into a single training cycle through the pruning while training approach.
- A robust pruning paradigm is suggested, which continuously monitors the evolving sub-structure guided by a pruning stability indicator to achieve an optimal architecture during the early stages of network training.
- The proposed pruning algorithm is validated on VGGNet, ResNet, and MobileNet architectures. Notably,

it achieves 75.49% top-1 and 92.63% top-5 accuracy for the ResNet50 model on the ImageNet dataset, with 1.38× faster training and 57% fewer FLOPs.

2. Related Work

Structured Pruning. While unstructured pruning has been a subject of long-standing research [20, 21, 29], structured pruning gained popularity with the emergence of modern DNNs [33]. Structured pruning revolves around extracting efficient sub-structures from complete models while optimizing storage and inference processes. This can involve incorporating regularization techniques to induce sparsity [43, 61] or defining criteria to identify and remove less crucial parameter groups [17, 25, 33], or a combination of both [14, 24, 55]. The practice of pruning at initialization [7, 15, 31], pruning during training [5, 6, 19, 51], and pruning the pre-trained model [9, 17, 54, 55, 60] also represents an extensively explored area in network pruning.

Regularization in Pruning. Regularization is a common approach to learning sparsity in DNNs [41, 43, 61]. The introduction of a penalty factor controls the level of regularization. When a modest penalty factor is uniformly applied to all weights, it facilitates the gradual acquisition of regular sparsity [14, 57]. Conversely, adopting a larger or progressively increasing penalty factor targeted at a specific parameter group yields large regularization effects [9, 54, 55]. Moreover, while utilizing parameter saliency is a widely adopted method for direct pruning [19, 33], the combination of parameter saliency and regularization is often used for the gradual learning of structural sparsity [9, 14, 54, 55]. While tuning task and model-specific hyperparameters has been challenging in regularization-based pruning, the advent of automatic parameter tuning algorithms using AutoML [13] and NAS [56] can simplify the process.

One-cycle Pruning. The pruning of a pre-trained network and fine-tuning the resulting pruned architecture is a commonly used framework for network slimming. However, this approach can be computationally intensive due to the multiple rounds of network training it entails. An alternative approach, known as pruning at initialization, quickly gained popularity and includes methods such as Lottery Ticket Hypothesis [15] and its variants [2, 59], single-shot network pruning (SNIP) [31], and progressive skeletonization [7]. However, some of its limitations are the risk of sub-optimal network structures, potential performance gaps, and difficulty in defining pruning criteria [15]. Another framework involving pruning and training in just a single training cycle includes methods such as PaT [51], LAASP [19], and only-train-once (OTO) [5, 6]. Although these approaches are more informative and have great potential compared to pruning at initialization, there is little attention in the literature. Therefore, in this paper, we further investigate the one-cycle pruning approach and develop an efficient structured pruning algorithm while achieving high accuracy.

3. Methods

3.1. Problem Formulation

Let us assume a convolutional neural network (CNN) with L layers, each layer parameterized by $\mathcal{W}^l \in \mathbb{R}^{C^l_{\text{out}} \times C^l_{\text{in}} \times K^l \times K^l}$, K being the kernel size and C_{out} and C_{in} being number of output and input channels, respectively. Given a dataset $\mathcal{D}$ consisting of N input-output pairs $\{(x_i, y_i)\}_{i=1}^{N}$, learning the network parameters with a given pruning ratio α can be formulated as the following optimization problem:

$$\arg\min_{\mathcal{M}} \big(\mathcal{L}(\mathcal{M}, \mathcal{D})\big), \quad \text{s.t.} \quad \frac{\Psi\Big(f(\mathcal{M}, x_i)\Big)}{\Psi\Big(f(\mathcal{W}, x_i)\Big)} \geq \alpha, \tag{1}$$

where $\mathcal{M} \subset \mathcal{W}$ are the parameters after pruning, $\mathcal{L}$ is the network loss, $f()$ encodes the network function, and $\Psi()$ maps the network parameters to the pruning constraints, i.e., in our case, float point operations (FLOPs) of the network. The method can easily be scaled to other constraints, such as latency or memory.

3.2. Group Saliency Criteria

Conventional saliency estimation for pruning convolutional filters considers only the weight values within this filter to measure its importance. However, in this paper, we also take into account its structurally associated coupled parameters for estimating the saliency, which we call group saliency. The DepGraph [14] algorithm is used to analyze parameter dependencies within W, and entire trainable parameters are partitioned into several groups $\mathcal{G} = \{g\}$. Simple norm-based criteria [33] is utilized for group saliency estimation. Each group $g = \{w_1, w_2, ..., w_{|g|}\}$ has an arbitrary number of sets of parameters with different dimensionality. Given our adoption of global pruning, we propose to incorporate ***local normalization*** into the calculation of group importance, thus facilitating the global ranking of filters for the pruning process. The overall saliency score of a group, as determined by the l_2-norm used in this study, is defined as:

$$S(g) = \frac{\sum_{w \in g} \|w\|_2 \,/ \sqrt{|w|}}{|g|}, \tag{2}$$

where $|.|$ denotes set cardinality.

In Eq. (2), we calculate the normalized l_2-norm [1] of the set of elements within a group. The resulting local norms are summed up and again normalized with the cardinality of the group. Such normalization is induced because the group size and dimensionality of the set of parameters within the groups could be different. This will ensure fair comparison within different groups or layers during global pruning.

3.3. Pruning Stability

During the pruning sub-network search process, at the end of each training epoch, we temporarily prune the network for a given pruning constraint and ratio using group saliency criteria defined in Eq. (2). Suppose $\mathcal{M}_{t-i}$ and $\mathcal{M}_t$ denotes the pruned sub-network structures at training epoch $t-i$ and t, respectively. Those sub-networks are used to check for pruning stability defined by their similarity. As soon as the sub-network stabilizes in subsequent training epochs, the pruning is made permanent, and the resulting pruned sub-network is further trained for convergence.

PaT [51] uses a similar approach, but they check for pruning stability calculated using only the number of channels in each layer. The limitation of their approach is that the two sub-networks could be identical in terms of the number of channels in each layer. However, the identity of those channels derived from the original network could still differ, leading to unstable pruning. To solve this issue, in this paper, we propose assessing pruning stability using the total number of retained filters and ***the identity of derived filters*** from the original network.

Suppose $\mathcal{F}^l_{t-i}$ and $\mathcal{F}^l_t$ be the set of filters in temporarily pruned sub-networks at training epoch $t-i$ and t, respectively. The similarity between these two sub-networks based on the cardinality of the filter set, as well as their corresponding identity, in each layer, is defined using Jaccard Similarity J:

$$J(\mathcal{M}_{t-i}, \mathcal{M}_t) = \frac{1}{L}\sum_{l=1}^{L}\frac{|\mathcal{F}^l_{t-i} \cap \mathcal{F}^l_t|}{|\mathcal{F}^l_{t-i} \cup \mathcal{F}^l_t|}. \tag{3}$$

The similarity value J ranges from 0 to 1, where 1 means two sub-networks are the same and vice-versa. In practice, we use the average of past r similarities:

$$J^t_{\text{avg}} = \frac{1}{r}\sum_{k=0}^{r-1} J(\mathcal{M}_{t-k-i}, \mathcal{M}_{t-k}). \tag{4}$$

The stability score determines when sparsity learning begins ($t_{\text{sl-start}}$) and is also used to identify the stable pruning epoch (t^*). $t_{\text{sl-start}}$ is found when consecutive epochs show minimal change in average similarity (J_{avg}) within a window (r), using a threshold value (τ). Therefore, the $t_{\text{sl-start}}$ is estimated using the following expression:

$$t_{\text{sl-start}} = \min\left\{t : \left(J^t_{\text{avg}} - J^{t-r}_{\text{avg}}\right) \leq \tau\right\}. \tag{5}$$

Additionally, the stable pruning epoch (t^*) is estimated using the following expression:

$$t^* = \arg\max_t \left(J^t_{\text{avg}} \geq 1 - \epsilon\right). \tag{6}$$

Here, t^* is determined as the epoch where the average similarity reaches a value greater than or equal to $(1-\epsilon)$, with ϵ representing the threshold value.

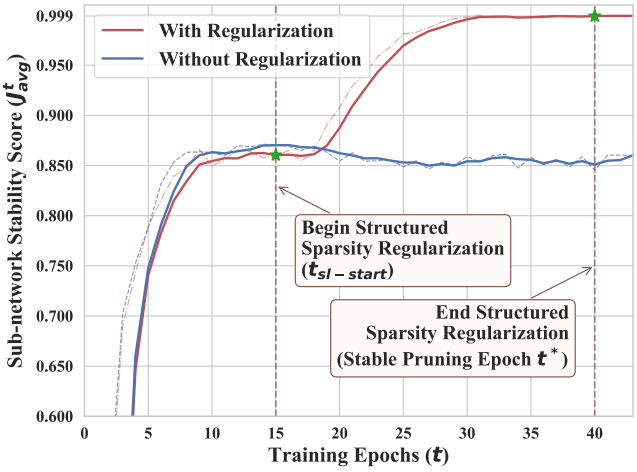

Figure 3. Sub-network stability score determined with Jaccard Similarity between sub-networks acquired during training epochs. The dotted curves are the plot of raw data before averaging.

Fig. 3 shows the curves illustrating the behavior of the sub-network stability score J^t_{avg} with and without the application of structured sparsity regularization. The figure highlights how regularization speeds up stability convergence, allowing sufficient time for fine-tuning the pruned architecture. The next section will explain this regularization technique in detail.

3.4. Structured Sparsity Regularization

The structured sparsity regularization scheme in our approach is motivated by pruning methods DepGraph [14] and GReg [55]. DepGraph [14] pruning scheme gradually sparsifies the structurally coupled parameters at the group level based on the normalized group importance scores. During the training process, the regularization is applied to all the parameter groups while driving the groups with low importance scores toward zero for pruning. On the other hand, GReg [55] imposes a penalty term on those filters selected for pruning during the training process while gradually increasing the regularization penalty. However, the GReg [55] technique ignores the structurally coupled parameters while imposing growing regularization for sparsity learning.

In this study, we also employ growing regularization to enhance sparsity learning, addressing structurally linked parameters at the group level. Algorithm 1 outlines the step-by-step procedure of the proposed pruning algorithm. In each training epoch, given group saliency scores and pruning ratio, global pruning is used to *temporarily partition* the structurally grouped parameters into *pruning and non-pruning groups* (Line 3 $\sim$ 5). The *binary search* algorithm is applied to quickly partition the groups for a given pruning ratio. In every training epoch, if a group's saliency score, which was previously partitioned as a pruning group, increases, it may move to the non-pruning group, and vice

Algorithm 1 Algorithm of OCSPruner

Input: Training data $\mathcal{D}$, base model with randomly initialized weights $\mathcal{W}$, pruning ratio α, total epochs T, window size for averaging stability scores r, sparsity learning start epoch threshold τ, pruning stability threshold ϵ

Output: Pruned model with fully trained weights $\mathcal{M}$

1: SL_STAGE $\leftarrow$ False
2: **for** $t = 0, \ldots, T-1$ **do**
3: Partition model $\mathcal{W}$ into structural groups $\mathcal{G}$
4: Get group saliency scores $S(g)$ using Eq. (2)
5: Obtain the pruned model $\mathcal{M}_t$ and pruning groups $\mathcal{G}^t_{\text{prune}}$ at pruning ratio α.
6: Get J^t_{avg} using Eq. (4)
7: **if** not SL_STAGE **then**
8: Train model $\mathcal{W}$ by gradient descent
9: **if** $\left(J^t_{\text{avg}} - J^{t-r}_{\text{avg}}\right) \leq \tau$ **then**
10: SL_STAGE $\leftarrow$ True
11: **end if**
12: **else if** $J^t_{\text{avg}} \geq 1 - \epsilon$ **then**
13: Set epoch t as stable pruning epoch, $t^* \leftarrow t$
14: $\mathcal{M} \leftarrow \mathcal{M}_{t^*}$
15: **break**
16: **else**
17: Update regularization penalty factor (λ_t) using Eq. (7)
18: Induce structural sparsity regularization using Eq. (8) and Eq. (9) on $\mathcal{W}$
19: Train model $\mathcal{W}$ by gradient descent
20: **end if**
21: **end for**
22: **for** $t = t^*, \ldots, T-1$ **do**
23: Train pruned model $\mathcal{M}$ by gradient descent
24: **end for**

versa. Then, our algorithm applies *regularization to the newly partitioned pruning groups*, leaving the non-pruning groups unaffected (Line 18).

In early network training, weights are volatile, thus, large regularization is not advisable. Therefore, initially, a small penalty is used for sparsity learning. As training advances the penalty is also increased to speed up regularization. The growing penalty factor used in our approach for structured sparsity regularization is defined as:

$$\lambda_t = \lambda_{t-1} + \delta \times \lfloor \frac{t - t_{\text{sl-start}}}{\Delta t} \rfloor. \tag{7}$$

In Eq. (7), t represents the training epoch, and δ denotes the growing factor. The term $t_{\text{sl-start}}$ corresponds to the starting epoch for structured sparsity regularization. From $t_{\text{sl-start}}$ onward, the penalty factor increases by δ in every Δt epoch interval, as determined by the floor function $\lfloor\ \rfloor$ (Line 17).

Suppose that, at training epoch t, $\mathcal{G}^t_{\text{prune}}$ and $\mathcal{G}^t_{\text{non-prune}}$ denote the partitioning of structural groups into pruning and non-pruning groups. First, similar to GReg [55], we impose a growing l_2-norm penalty estimated using a pruning group of parameters to the network's original loss function defined as:

$$\mathcal{L}^t_{\text{total}} = \mathcal{L}^t_{\text{orig}} + \lambda_t \sum_{g \in \mathcal{G}^t_{\text{prune}}} \sum_{w \in g} \|w\|_2 , \tag{8}$$

where $\mathcal{L}^t_{\text{orig}}$ is the network original loss before imposing penalty term.

The structured sparsity regularization imposed by Eq. (8) gradually drives the pruning group parameters to zero. But, as our algorithm intends to finalize the pruning process as early as possible, we define an additional sparsity learning scheme to further drive the pruning group of parameters to zero. To achieve this, we propose to directly multiply pruning group parameters with a multiplication factor estimated using the network's current learning rate and growing penalty factor as follows:

$$\mathcal{G}^t_{\text{prune}} = \{g(1 - \lambda_t \times \text{learning-rate}_t) \mid g \in \mathcal{G}^t_{\text{prune}}\}. \tag{9}$$

In the early stage, sparsity learning, Eq. (9) gently reduces weights, then gradually intensifies. This two-step process provides smooth yet fast regularization, making pruning stable early while leaving enough time for fine-tuning the pruned model.

4. Results and Discussion

4.1. Experimental Setup

To evaluate our pruning algorithm, we conducted experiments on the CIFAR-10/100 [28] and ImageNet [48] datasets. Initially, we pruned the relatively simple single-branch VGGNet [52] on the CIFAR-10 and CIFAR-100 datasets. Next, we evaluate the pruning of more complex multi-branch ResNet [22] models, on both CIFAR-10 and ImageNet. Finally, we evaluated the pruning of a compact network, MobileNetV2 [49], on the ImageNet.

Both CIFAR and ImageNet datasets are trained with a Stochastic Gradient Descend (SGD) optimizer with a momentum of 0.9 using data argumentation strategies adapted from official PyTorch [47] examples. For training on CIFAR-10/100 datasets, the models are trained for 300 epochs with batch size 128, in which the learning rate follows the MultiStepLR scheduler. Conversely, for training on the ImageNet dataset, the learning rate follows a cosine function-shaped decay strategy, gradually approaching zero. The training settings for MobileNetV2 on the ImageNet dataset are adapted from HBONet [32]. Please refer to the supplement for more comprehensive information on the training and pruning parameters.

For the CIFAR dataset, we observe slight variations across training trials; therefore, we report the average over three runs. In contrast, ImageNet results were consistent across runs; hence, results from a single trial are reported.

Table 1. Pruning of VGG16/19 on CIFAR-10/100 dataset. Our results are the average outcome from three trials.

Model/ Dataset	Method	Pre-train?	FLOPs (%)	Params (%)	Base Acc. (%)	Pruned Acc. (%)	Acc. Drop (%)
VGG16/ CIFAR-10	RCP [35]	✓	47.99	42.11	94.33	93.94	0.39
	PGMPF [4]	✓	34.00	-	93.68	93.60	0.08
	CPGCN [27]	✓	26.93	6.94	93.10	93.08	0.02
	LAASP [19]	×	39.54	26.83	93.79	93.79	0.00
	OTOv1 [5]	×	26.80	5.50	93.20	93.30	-0.10
	OCSPruner	×	26.01	9.72	94.08	93.88	0.20
	DLRFC [26]	✓	23.05	5.62	93.25	93.64	-0.39
	OTOv2 [6]	×	23.70	4.90	93.20	93.20	0.00
	DCFF [40]	×	23.13	7.20	93.02	93.47	-0.45
	OCSPruner	×	21.22	7.46	94.08	93.76	0.32
VGG19/ CIFAR-100	ED [53]	✓	11.37	-	73.35	65.18	8.17
	GReg-2 [55]	✓	11.31	-	74.02	67.75	6.27
	DepGraph [14]	✓	11.21	-	73.50	70.39	3.11
	OCSPruner	×	11.23	5.49	74.24	70.47	3.77

Table 2. Pruning ResNets on CIFAR-10 dataset. Our results are the average outcome from three trials.

Depth	Method	Pre-train?	FLOPs (%)	Params (%)	Base Acc. (%)	Pruned Acc. (%)	Acc. Drop (%)
56	HRank [37]	✓	50.00	-	93.26	93.17	0.09
	DLRFC [26]	✓	47.42	44.37	93.06	93.57	-0.51
	ResRep [12]	✓	47.09	-	93.71	93.71	0.00
	NuSPM [30]	×	49.72	-	-	93.50	-
	FPGM [24]	×	47.40	-	93.59	92.93	0.66
	LAASP [19]	×	47.40	61.43	93.61	93.45	0.16
	OCSPruner	×	46.93	50.27	93.97	93.80	0.17
	GReg-2 [55]	✓	39.21	-	93.36	93.36	0.00
	C-SGD [10]	✓	39.15	-	93.39	93.44	-0.05
	DepGraph [14]	✓	38.91	-	93.53	93.64	-0.11
	ATO [58]	×	45.00	-	93.50	93.74	-0.24
	DCFF [40]	×	43.75	44.71	93.26	93.26	0.00
	OCSPruner	×	38.88	41.42	93.97	93.65	0.32
110	FPGM [24]	×	47.70	-	93.68	93.74	-0.06
	OCSPruner	×	46.37	52.71	94.36	94.30	0.06
	HRank [37]	✓	41.80	-	93.50	93.36	0.14
	LAASP [19]	×	41.29	54.56	94.41	93.61	0.80
	DCFF [40]	×	33.18	32.37	93.50	93.80	-0.30
	OCSPruner	×	33.10	34.93	94.36	94.13	0.23

4.2. VGG16/19 on CIFAR-10/100

Tab. 1 compares the pruning of VGG16/19 on CIFAR-10/100 datasets using our method against several state-of-the-art approaches. The VGG16 pruning on CIFAR-10 is compared with RCP [35], LAASP [19], PGMPF [4], CPGCN [27], OTO [5, 6], DLRFC [26], and DCFF [40]. Among these methods, similar to ours, OTO, and DCFF also pruned the network from scratch. While OTO, CPGCN, and DLRFC achieve higher parameter reduction rates, their top-1 accuracy is lower than ours. Notably, OTO uses a specially designed training optimizer, whereas our method works with the standard SGD optimizer. With only 26.01% and 21.22% of the original network FLOPs remaining, we achieved top-1 accuracy of 93.88% and 93.76%, respectively, the highest reported for similar pruning rates. Notably, while DLRFC and DCFF show improvements in pruned model performance, this is largely due to their comparison against relatively lower baseline accuracy.

Tab. 1 also shows our method's VGG19 pruning performance in comparison with EigenDamage [53], GReg [55], and DepGraph [14] on the CIFAR-100 dataset. We achieved the highest top-1 accuracy of 70.47% while reducing the network parameters around 95% and FLOPs around 89%. Although GReg [55] also utilizes the concept of growing regularization and uses a fully trained network for pruning, its top-1 accuracy is nearly 3% lower than ours.

4.3. ResNet on CIFAR-10

The pruning of ResNet on CIFAR-10 using our method consistently outperforms state-of-the-art methods across depths 56 and 110 (Tab. 2). While our proposed algorithm automatically learns slimmer sub-networks from scratch, we not only compare its performance with other automatic methods like FPGM [24], and LAASP [19] but also assess its effectiveness against various other state-of-the-art techniques such as DLRFC [26], HRank [37], ResRep [12], ATO [58], DCFF [40], DepGraph [14], GReg [55], C-SGD [10], and Rethink [42]. Similar to ours, LAASP [19] employs a partially trained network for pruning. However, it ignores structurally coupled parameters while estimating filter saliency scores and adapts manually defined stable pruning epochs. In contrast to our technique, which gradually moves pruning parameters towards zero, FPGM [24] directly sets filters with low saliency scores to zero, while still allowing updates in the next training epoch. We prune ResNet56 and ResNet110 models for two different FLOPs reduction rates. Notably, with 38.88% reduced FLOPs, ResNet56 achieves 93.65% top-1 accuracy. Again, ResNet110 achieves a remarkable 94.13% top-1 accuracy with only 33.10% FLOPs.

4.4. ResNet50/MobileNetV2 on ImageNet

We evaluate the performance of ResNet50 and MobileNetV2 on the ImageNet dataset across different pruning ratios, comparing our results with those of various state-of-the-art methods from the literature (Tab. 3). Our method consistently surpasses the performance of several state-of-the-art approaches. Notably, similar to our approach, FPGM [24], PaT [51], NuSPM [30], and OTOv1 [5] also achieve a fully trained pruned network using one-cycle training from scratch. Remarkably, with a mere 43% of the

Table 3. Pruning ResNet50/MobileNetV2 on ImageNet dataset.

Model	Method	Pre-train?	FLOPs (%)	Params (%)	Baseline Top-1 Acc. (%)	Pruned Top-1 Acc. (%)	Top-1 Acc. Drop (%)	Baseline Top-5 Acc. (%)	Pruned Top-5 Acc. (%)	Top-5 Acc. Drop (%)	Training SpeedUp
ResNet50	CLR-RNF [38]	✓	59.60	66.20	76.01	74.85	1.16	92.96	92.31	0.65	$\leq 0.66\times$
	FPGM [24]	×	57.80	-	76.15	75.03	1.12	92.87	92.40	0.47	$\leq 1.00\times$
	LAASP [19]	×	57.67	77.29	76.48	75.85	0.63	93.14	92.81	0.33	$1.04\times$
	OCSPruner$_{43\%}$	×	56.99	64.45	76.29	76.00	0.29	93.04	92.90	0.14	$1.28\times$
	HRank [37]	✓	56.18	-	76.15	74.98	1.17	92.87	92.33	0.54	$< 0.50\times$
	Tayler [44]	✓	54.95	-	76.18	74.50	1.68	-	-	-	$< 0.50\times$
	White-Box [64]	✓	54.34	68.63	76.15	75.32	0.83	92.96	92.43	0.53	$\leq 0.66\times$
	DepGraph [14]	✓	51.82	-	76.15	75.83	0.32	-	-	-	$< 0.50\times$
	OCSPruner$_{48\%}$	×	51.74	62.66	76.29	75.95	0.34	93.04	92.75	0.29	$1.30\times$
	GNN-RL [62]	✓	47.00	-	76.10	74.28	1.82	-	-	-	$\leq 0.66\times$
	PGMPF [4]	✓	46.50	-	76.15	75.11	0.90	92.93	92.41	0.52	$\leq 0.66\times$
	GReg-2 [55]	✓	43.33	62.30	76.13	75.36	0.77	92.86	92.41	0.45	$\leq 0.66\times$
	PaT [51]	×	41.30	-	-	74.85	-	-	-	-	$> 1.00\times$
	NuSPM [30]	×	41.30	-	-	75.30	-	-	-	-	$> 1.00\times$
	OCSPruner$_{57\%}$	×	42.98	51.83	76.29	75.49	0.80	93.04	92.63	0.41	$1.38\times$
	CC [34]	✓	37.32	41.39	76.15	74.54	1.61	92.87	92.25	0.62	$\leq 0.66\times$
	DTP [36]	✓	32.68	-	76.13	74.26	1.87	-	-	-	$0.66\times$
	OTOv1 [5]	×	34.50	35.50	76.10	74.70	1.40	92.90	92.10	0.80	$< 1.00\times$
	OCSPruner$_{66\%}$	×	33.05	37.13	76.29	74.72	1.57	93.04	92.13	0.91	$1.41\times$
MobileNetV2	CC [34]	✓	71.67	-	71.88	70.91	0.97	-	-	-	$\leq 0.66\times$
	AMC [23]	×	70.00	-	71.80	70.80	1.00	-	-	-	-
	OCSPruner$_{30\%}$	×	70.32	87.88	71.79	70.96	0.83	90.47	89.84	0.63	$1.20\times$
	DepGraph [14]	✓	45.45	-	71.87	68.46	3.41	-	-	-	$< 0.50\times$
	DCFF [40]	×	46.67	74.86	72.00	68.60	3.40	90.12	88.13	1.99	$> 1.00\times$
	LAASP [19]	×	45.50	64.09	71.79	68.45	3.34	90.47	88.40	2.07	$1.08\times$
	OCSPruner$_{55\%}$	×	45.40	67.76	71.79	68.68	3.11	90.47	88.16	2.31	$1.20\times$

original FLOPs remaining in the network, our algorithm achieves the highest accuracy (75.49% top-1 and 92.63% top-5) among state-of-the-art methods in the literature for similar pruning ratios. Fig. 1 graphically illustrates the relationship between network FLOPs and top-1 accuracy, providing an approximation of the training speed-up associated with each pruning method in reference to the baseline network training. This clearly shows that our proposed technique consistently achieves higher top-1 accuracy compared to several other state-of-the-art methods while also being the most efficient pruning framework among them.

Tab. 3 also presents the pruning outcomes for MobileNetV2 on ImageNet and compares them with several state-of-the-art methods, including CC [34], AMC [23], DCFF [40], LAASP [19], and DepGraph [14]. We report results using the standard $1.0\times$ MobileNetV2 for fair comparison. Experiments are conducted at 30% and 55% pruning ratios to align with commonly used benchmarks. In both instances, our proposed method attains superior top-1 accuracy compared to the other methods.

The last column in Tab. 3 presents the comparison results for training speed enhancement (higher the better) relative to the baseline ResNet50 model training cost. As shown, our proposed technique achieved up to a $1.41\times$ increase in training speed while pruning 67% of ResNet50 FLOPs. We approximated training speedup for state-of-the-art methods based on factors such as the total number of training rounds and whether pruning was done iteratively or in one shot. This comparative speedup analysis demonstrates that our method is the most efficient among the existing techniques.

4.5. Ablation Studies

Correlation between Sub-network Stability Score and final Accuracy. Fig. 4 illustrates the correlation between the stability score and the final accuracy of models pruned at various epochs until the training process reaches a stable pruning epoch. We measured the final accuracy of each temporarily pruned model obtained during the optimal sub-network search process guided by the pruning stability indicator. The Fig. 4 clearly shows that as the stability score approaches its highest value, the best-performing pruned sub-network is also selected. The Fig. 4 also shows that if we further delay the pruning process even after the stability score has peaked, the final accuracy of the pruned network begins to decline. This occurs because, once the pruning has

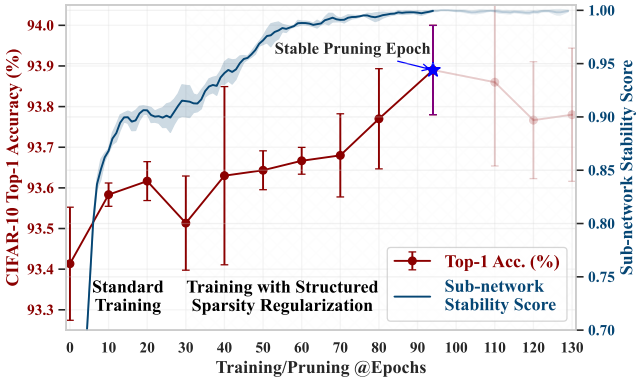

Figure 4. Tracking the sub-network stability score and the corresponding pruned model's final accuracy over epochs for ResNet56 pruning on CIFAR-10 with a 50% FLOPs reduction rate.

Table 4. The influence on varying Sparsity Learning (SL) start epoch ($t_{\text{sl-start}}$) while pruning ResNet50 on the ImageNet. Given $t_{\text{sl-start}}$, t^*, i.e., stable pruning epoch, was identified using Eq. (6).

SL Start Epoch ($t_{\text{sl-start}}$)	SL End Epoch (t^*)	Total SL Epochs ($t^* - t_{\text{sl-start}}$)	Top-1 Acc. (%)	Training Speed Up
0	26	26	75.54	1.28×
15	31	16	75.90	1.28×
30	53	23	75.93	1.19×
45	69	24	75.81	1.13×
60	86	26	75.67	1.08×
17†	36	19	**75.95**	**1.30**×

† Automatically estimated using Eq. (5); all other *SL Start Epoch* values were manually assigned.

stabilized, the structure of the pruned sub-network hardly changes, and there will be fewer and fewer training epochs left for fine-tuning the pruned network.

Influence of Varying Sparsity Learning Start Epoch ($t_{\text{sl-start}}$). The start epoch for sparsity learning significantly influences the proposed method's overall training cost and performance. The algorithm behavior concerning training speed and performance for different values of $t_{\text{sl-start}}$ is illustrated in Tab. 4. After a few initial training epochs, superior results were achieved by applying sparsity learning. Once the proposed stability indicator shows minimal variation across consecutive epochs, it indicates an optimal time to commence sparsity learning, as indicated by Eq. (5). The Tab. 4 shows that delaying this process only leads to performance degradation along with increased training time.

Pruning Pre-trained Models. While OCSPruner is primarily designed for pruning from scratch, we also applied it to pre-trained models to reflect practical scenarios. Using the same training and pruning settings, we replaced randomly initialized models with pre-trained ones. As shown in Tab. 5, OCSPruner achieves significant compression on

Table 5. Pruning pre-trained models with OCSPruner.

Dataset	Model	FLOPs (%)	Params (%)	Base Top1 Acc. (%)	Pruned Top1 Acc. (%)	Top1 Acc. ↓ (%)
CIFAR-10	VGG16	21.21	13.68	94.07	93.63	0.44
	ResNet56	38.82	42.26	94.01	93.50	0.51
CIFAR-100	VGG19	11.22	10.06	73.58	69.98	3.64
ImageNet	ResNet50	51.73	56.05	76.29	76.22	0.07

VGG16 and ResNet56 (CIFAR-10) and ResNet50 (ImageNet), with minimal accuracy loss. This shows its effectiveness for randomly initialized and pre-trained models, with potential for further gains via hyperparameter tuning.

5. Conclusion

We presented OCSPruner, a novel one-cycle structured pruning framework that integrates training and pruning into a single efficient process. By introducing a stability-driven sub-network search guided by a pruning stability indicator, our method enables early and effective pruning. Through structured sparsity regularization and adaptive penalty scheduling, OCSPruner ensures both stable pruning and high model accuracy. Experiments across CIFAR and ImageNet benchmarks demonstrate that OCSPruner achieves state-of-the-art performance in terms of accuracy and training efficiency, outperforming several competitive methods. Moreover, it remains versatile and effective for both from-scratch and pre-trained models. Future work will investigate alternative saliency metrics and broader applications to modern architectures and other domains.

Limitation. Although our method removes redundant structures via regularization, it requires careful, task- and architecture-specific hyperparameter tuning. The algorithm is best suited for training schedules with gradual changes (e.g., cosine or linear decay), as they enable stable pruning by avoiding abrupt shifts in learning rates.

Acknowledgment. This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2025-16071992), the convergence security core talent training business support program(IITP-2024 2024-RS-2024-00426853) supervised by the IITP(Institute of Information & Communications Technology Planning & Evaluation), and G-LAMP Program of the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education (No. RS-2025-25441317). Additionally, this work was also supported by the Technology Innovation Program (20018906, Development of autonomous driving collaboration control platform for commercial and task assistance vehicles) funded by the Ministry of Trade, Industry and Energy (MOTIE, Korea).

References

[1] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *NeurIPS*, 2016. 3

[2] Yue Bai, Huan Wang, ZHIQIANG TAO, Kunpeng Li, and Yun Fu. Dual Lottery Ticket Hypothesis. In *ICLR*, 2022. 3

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 33:1877–1901, 2020. 1

[4] Linhang Cai, Zhulin An, Chuanguang Yang, Yangchun Yan, and Yongjun Xu. Prior gradient mask guided pruning-aware fine-tuning. *AAAI*, 36(1):140–148, 2022. 6, 7

[5] Tianyi Chen, Bo Ji, DING Tianyu, Biyi Fang, Guanyi Wang, Zhihui Zhu, Luming Liang, Yixin Shi, Sheng Yi, and Xiao Tu. Only Train Once: A one-shot neural network training and pruning framework. In *NIPS*, 2021. 1, 2, 3, 6, 7

[6] Tianyi Chen, Luming Liang, DING Tianyu, Zhihui Zhu, and Ilya Zharkov. OTOv2: Automatic, generic, user-friendly. In *ICLR*, 2023. 1, 3, 6

[7] Pau de Jorge, Amartya Sanyal, Harkirat Behl, Philip Torr, Grégory Rogez, and Puneet K. Dokania. Progressive skeletonization: Trimming more fat from a network at initialization. In *ICLR*, 2021. 2, 3

[8] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *ProcIEEE*, 108(4):485–532, 2020. 1

[9] Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *AAAI*, 2018. 3

[10] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal SGD for pruning very deep convolutional networks with complicated structure. In *CVPR*, pages 4943–4953, 2019. 6

[11] Xiaohan Ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, Ji Liu, et al. Global sparse momentum SGD for pruning very deep neural networks. *NeurIPS*, 32, 2019. 1

[12] Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. ResRep: Lossless cnn pruning via decoupling remembering and forgetting. In *ICCV*, pages 4510–4520, 2021. 6

[13] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *ICML*, pages 1437–1446. PMLR, 2018. 3

[14] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. DepGraph: Towards any structural pruning. In *CVPR*, pages 16091–16101, 2023. 1, 2, 3, 4, 6, 7

[15] Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019. 1, 2, 3

[16] Amin Ghafourian, Zhongying CuiZhu, Debo Shi, Ian Chuang, Francois Charette, Rithik Sachdeva, and Iman Soltani. Hierarchical end-to-end autonomous navigation through few-shot waypoint detection. *IEEE Robot. Autom. Lett.*, 2024. 2

[17] Deepak Ghimire and Seong-Heum Kim. Magnitude and similarity based variable rate filter pruning for efficient convolution neural networks. *Applied Sciences*, 13(1):316, 2022. 3

[18] Deepak Ghimire, Dayoung Kil, and Seong-heum Kim. A survey on efficient convolutional neural networks and hardware acceleration. *Electronics*, 11(6):945, 2022. 1

[19] Deepak Ghimire, Kilho Lee, and Seong-heum Kim. Loss-aware automatic selection of structured pruning criteria for deep neural network acceleration. *IVC*, 136:104745, 2023. 2, 3, 6, 7

[20] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *NeurIPS*, 28, 2015. 1, 3

[21] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *NeurIPS*. Morgan-Kaufmann, 1992. 3

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1, 5

[23] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. In *ECCV*, pages 784–800, 2018. 7

[24] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*, pages 4340–4349, 2019. 3, 6, 7

[25] Yang He, Ping Liu, Linchao Zhu, and Yi Yang. Filter pruning by switching to neighboring CNNs with good attributes. *TNNLS*, 2022. 3

[26] Zhiqiang He, Yaguan Qian, Yuqi Wang, Bin Wang, Xiaohui Guan, Zhaoquan Gu, Xiang Ling, Shaoning Zeng, Haijiang Wang, and Wujie Zhou. Filter pruning via feature discrimination in deep neural networks. In *ECCV*, pages 245–261. Springer, 2022. 6

[27] Di Jiang, Yuan Cao, and Qiang Yang. On the channel pruning using graph convolution network for convolutional neural network acceleration. In *IJCAI*, pages 3107–3113. International Joint Conferences on Artificial Intelligence Organization, 2022. Main Track. 6

[28] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009. 5

[29] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In *NeurIPS*. Morgan-Kaufmann, 1989. 3

[30] Donghyeon Lee, Eunho Lee, and Youngbae Hwang. Pruning from scratch via shared pruning module and nuclear norm-based regularization. In *WACV*, pages 1393–1402, 2024. 6, 7

[31] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: single-shot network pruning based on connection sensitivity. In *ICLR*, 2019. 2, 3

[32] Duo Li, Aojun Zhou, and Anbang Yao. HBONet: Harmonious bottleneck on two orthogonal dimensions. In *ICCV*, pages 3316–3325, 2019. 5

[33] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017. 1, 3

[34] Yuchao Li, Shaohui Lin, Jianzhuang Liu, Qixiang Ye, Mengdi Wang, Fei Chao, Fan Yang, Jincheng Ma, Qi Tian, and Rongrong Ji. Towards compact CNNs via collaborative compression. In *CVPR*, pages 6438–6447, 2021. 7

[35] Yawei Li, Kamil Adamczewski, Wen Li, Shuhang Gu, Radu Timofte, and Luc Van Gool. Revisiting random channel pruning for neural network compression. In *CVPR*, pages 191–201, 2022. 6

[36] Yunqiang Li, Jan C van Gemert, Torsten Hoefler, Bert Moons, Evangelos Eleftheriou, and Bram-Ernst Verhoef. Differentiable transportation pruning. In *ICCV*, pages 16957–16967, 2023. 7

[37] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. HRank: Filter pruning using high-rank feature map. In *CVPR*, pages 1529–1538, 2020. 6, 7

[38] Mingbao Lin, Liujuan Cao, Yuxin Zhang, Ling Shao, Chia-Wen Lin, and Rongrong Ji. Pruning networks with cross-layer ranking & k-reciprocal nearest filters. *TNNLS*, 2022. 7

[39] Mingbao Lin, Yuxin Zhang, Yuchao Li, Bohong Chen, Fei Chao, Mengdi Wang, Shen Li, Yonghong Tian, and Rongrong Ji. 1xN pattern for pruning convolutional neural networks. *PAMI*, 45(4):3999–4008, 2022. 1

[40] Mingbao Lin, Bohong Chen, Fei Chao, and Rongrong Ji. Training compact CNNs for image classification using dynamic-coded filter fusion. *PAMI*, 2023. 6, 7

[41] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, pages 2736–2744, 2017. 3

[42] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019. 6

[43] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *ICLR*, 2018. 3

[44] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *CVPR*, pages 11264–11272, 2019. 7

[45] OpenAI. GPT-4 technical report, 2023. 1

[46] Cheonjun Park, Mincheol Park, Hyun Jae Oh, Minkyu Kim, Myung Kuk Yoon, Suhyun Kim, and Won Woo Ro. Balanced column-wise block pruning for maximizing gpu parallelism. In *AAAI*, pages 9398–9407, 2023. 1

[47] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017. 5

[48] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115:211–252, 2015. 5

[49] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018. 5

[50] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *NeurIPS*, 33:20378–20389, 2020. 1

[51] Maying Shen, Pavlo Molchanov, Hongxu Yin, and Jose M. Alvarez. When To Prune? a policy towards early structural pruning. In *CVPR*, pages 12247–12256, 2022. 1, 2, 3, 4, 6, 7

[52] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 5

[53] Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. EigenDamage: Structured pruning in the kronecker-factored eigenbasis. In *ICML*, pages 6566–6575. PMLR, 2019. 6

[54] Huan Wang, Qiming Zhang, Yuehai Wang, Lu Yu, and Haoji Hu. Structured pruning for efficient convnets via incremental regularization. In *IJCNN*, pages 1–8. IEEE, 2019. 3

[55] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. In *ICLR*, 2021. 1, 3, 4, 5, 6, 7

[56] Haibin Wang, Ce Ge, Hesen Chen, and Xiuyu Sun. PreNAS: Preferred one-shot learning towards efficient neural architecture search. In *ICML*, pages 35642–35654. PMLR, 2023. 3

[57] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *NeurIPS*, 29, 2016. 3

[58] Xidong Wu, Shangqian Gao, Zeyu Zhang, Zhenzhen Li, Runxue Bao, Yanfu Zhang, Xiaoqian Wang, and Heng Huang. Auto-Train-Once: Controller network guided automatic network pruning from scratch. In *CVPR*, pages 16163–16173, 2024. 6

[59] Zheyang Xiong, Fangshuo Liao, and Anastasios Kyrillidis. Strong lottery ticket hypothesis with ε–perturbation. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, pages 6879–6902. PMLR, 2023. 3

[60] Hancheng Ye, Bo Zhang, Tao Chen, Jiayuan Fan, and Bin Wang. Performance-aware approximation of global channel pruning for multitask CNNs. *PAMI*, 2023. 3

[61] Jianbo Ye, Xin Lu, Zhe Lin, and James Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *ICLR*, 2018. 3

[62] Sixing Yu, Arya Mazaheri, and Ali Jannesari. Topology-aware network pruning using multi-stage graph embedding and reinforcement learning. In *ICML*, pages 25656–25667. PMLR, 2022. 7

[63] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *CVPR*, pages 12104–12113, 2022. 1

[64] Yuxin Zhang, Mingbao Lin, Chia-Wen Lin, Jie Chen, Yongjian Wu, Yonghong Tian, and Rongrong Ji. Carrying out CNN channel pruning in a white box. *TNNLS*, 2022. 7

[65] Yuxin Zhang, Mingbao Lin, ZhiHang Lin, Yiting Luo, Ke Li, Fei Chao, YONGJIAN WU, and Rongrong Ji. Learning best combination for efficient N:M sparsity. In *NeurIPS*, 2022. 1