DepGraph: Towards Any Structural Pruning

Gongfan Fang[1]   Xinyin Ma[1]   Mingli Song[2]   Michael Bi Mi[3]   Xinchao Wang[1*]

National University of Singapore[1]   Zhejiang University[2]   Huawei Technologies Ltd.[3]

gongfan@u.nus.edu, maxinyin@u.nus.edu, xinchao@nus.edu.sg

https://github.com/VainF/Torch-Pruning

Abstract

Structural pruning enables model acceleration by removing structurally-grouped parameters from neural networks. However, the parameter-grouping patterns vary widely across different models, making architecture-specific pruners, which rely on manually-designed grouping schemes, non-generalizable to new architectures. In this work, we study a highly-challenging yet barely-explored task, any structural pruning, to tackle general structural pruning of arbitrary architecture like CNNs, RNNs, GNNs and Transformers. The most prominent obstacle towards this goal lies in the structural coupling, which not only forces different layers to be pruned simultaneously, but also expects all removed parameters to be consistently unimportant, thereby avoiding structural issues and significant performance degradation after pruning. To address this problem, we propose a general and fully automatic method, Dependency Graph (DepGraph), to explicitly model the dependency between layers and comprehensively group coupled parameters for pruning. In this work, we extensively evaluate our method on several architectures and tasks, including ResNe(X)t, DenseNet, MobileNet and Vision transformer for images, GAT for graph, DGCNN for 3D point cloud, alongside LSTM for language, and demonstrate that, even with a simple norm-based criterion, the proposed method consistently yields gratifying performances.

1. Introduction

The recent emergence of edge computing applications calls for the necessity for deep neural compression [16, 22, 25, 33, 34, 61, 65–67, 69, 75]. Among the many network compression paradigms, pruning has proven itself to be highly effective and practical [7, 11, 30, 31, 44, 58, 59, 74]. The goal of network pruning is to remove redundant parameters from a given network to lighten its size and potentially speed up the inference. Mainstream pruning approaches can
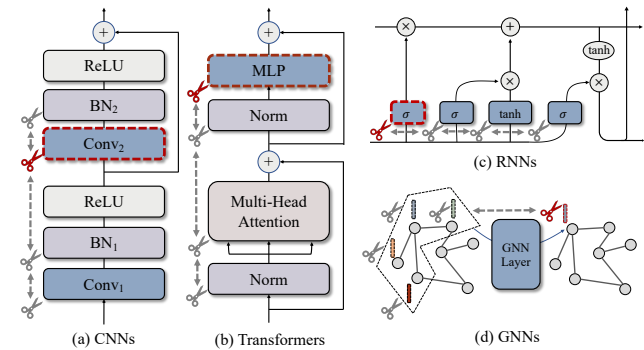
*Corresponding author

Figure 1. Parameters from different layers are inherently dependent on each other across network architectures, which forces several layers to be pruned simultaneously. For instance, to prune the $Conv_2$ in (a), all other layers $\{Conv_1, BN_1, BN_2\}$ within the block must be pruned as well. In this work, we introduce a generic scheme, termed as Dependency Graph, to explicitly account for such dependencies and execute the pruning of arbitrary architecture in a fully automatic manner.

be roughly categorized into two schemes, *structurual pruning* [4, 29, 71] and *unstructurual pruning* [8, 13, 44]. The core difference between the two lies in that, structural pruning changes the structure of neural networks by physically removing grouped parameters, while unstructural pruning conducts zeroing on partial weights without modification to the network structure. Compared to unstructural ones, structural pruning does not rely on specific AI accelerators or software to reduce memory consumption and computational costs, thereby finding a wider domain of applications in practice [38, 68].

Nevertheless, the nature of structural pruning *per se* makes itself a challenging task, especially for modern deep neural networks with coupled and complex internal structures. The rationale lies in that, deep neural networks are built upon a large number of basic modules like convolution, normalization, or activation, yet these modules, either parameterized or not, are intrinsically coupled through the intricate connections [17, 23]. As a result, even when we seek to remove only one channel from a CNN illustrated in

Figure 1(a), we have to take care of its inter-dependencies to all layers simultaneously, otherwise we will eventually get a broken network. To be exact, the residual connection requires the output of two convolutional layers to share the same number of channels and thus forces them to be pruned together [20, 40, 71]. The same goes for structural pruning on other architectures like Transformers, RNNs and GNNs as illustrated in Figs. 1(b-d).

Unfortunately, dependency does not only emerge in residual structures, which can be infinitely complex in modern models [23, 46]. Existing structural approaches have largely relied on case-by-case analyses to handle dependencies in networks [29, 40]. Despite the promising results achieved, such a network-specific pruning approach is effort-consuming. Moreover, these methods are not directly generalizable, meaning that the manually-designed grouping scheme is not transferable to other network families or even the network architectures in the same family, which in turn, greatly limit their industrial applications.

In this paper, we strive for a generic scheme towards *any structural pruning*, where structural pruning over arbitrary network architectures is executed in an automatic fashion, At the heart of our approach is to estimate the *Dependency Graph* (DepGraph), which explicitly models the interdependency between paired layers in neural networks. Our motivation to introduce DepGraph for structural pruning stems from the observation that, structural pruning at one layer effectively "triggers" pruning at adjacent layers, which further leads to a chain effect like $\{BN_2 \leftarrow Conv_2 \rightarrow BN_1 \rightarrow Conv_1\}$ as shown in Figure 1(a). As such, to trace the dependencies across different layers, we may decompose and model the dependency chain as a recursive process, which naturally boils down to the problem of finding the maximum connected components in the graph, and can be solved in $O(N)$ complexity via graph traversal.

It is also worth noting that in structural pruning, grouped layers are pruned simultaneously, which expects all removed parameters in the same group to be consistently unimportant. This brings certain difficulties to existing importance criteria designed for a single layer [20, 27, 29, 42]. To be exact, the parameter importance in a single layer no longer reveals correct importance due to the entanglement with other parameterized layers. To address this problem, we fully leverage the comprehensive ability of dependency modeling powered by DepGraph to design a "group-level" importance criterion, which learns consistent sparsity within groups, so that those zeroized groups can be safely removed without too much performance degradation.

To validate the effectiveness of DepGraph, we apply the proposed method to several popular architectures including CNNs [23, 40], Transformers [10], RNNs [12, 53], and GNNs [55], where competitive performance is achieved compared to state-of-the-art methods [7, 32, 58, 71]. For CNN pruning, our method obtains a 2.57× accelerated ResNet-56 model with 93.64% accuracy on CIFAR, which is even superior to the unpruned model with 93.53% accuracy. And on ImageNet-1k, our algorithm achieves more than 2× speed-up on ResNet-50, with only 0.32% performance lost. *More importantly*, our method can be readily transferred to various popular networks, including ResNe(X)t [40, 63], DenseNet [23], VGG [51], MobileNet [48], GoogleNet [54] and Vision Transformer [10], and demonstrate gratifying results. Besides, we also conduct further experiments on non-image neural networks, including LSTM [12] for text classification, DGCNN [60] for 3D point cloud, and GAT [55] for graph data, where our method achieves from 8× to 16× acceleration without a significant performance drop.

In sum, our contribution is a generic pruning scheme towards any structural pruning, termed as Dependency Graph (DepGraph), which allows for automatic parameter grouping and effectively improves the generalizability of structural pruning over various network architectures, including CNNs, RNNs, GNNs and Vision Transformers.

2. Related Work

Structural and Unstructural Pruning. Pruning has made tremendous progress in the field of network acceleration, as evidenced by various studies in the literature [2, 19–21, 29, 35, 39]. Mainstream pruning methods can be broadly categorized into two types: structural pruning [4, 29, 32, 71, 71] and unstructural pruning [8, 28, 44, 49]. Structural pruning aims to physically remove a group of parameters, thereby reducing the size of neural networks. In contrast, unstructured pruning involves zeroing out specific weights without altering the network structure. In practice, unstructural pruning, in particular, is straightforward to implement and inherently adaptable to various networks. however, it often necessitates specialized AI accelerators or software for model acceleration [15]. Conversely, structural pruning improves the inference overhead by physically removing parameters from networks, thereby finding a wider domain of applications [29, 38]. In the literature, The design space of pruning algorithms encompasses a range of aspects, including pruning schemes [21, 39], parameter selection [20, 43, 44], layer sparsity [27, 49] and training techniques [47, 58]. In recent years, numerous robust criteria have been introduced, such as magnitude-based criteria [20, 27, 70] and gradient-based criteria [32, 37] and learned sparsity [7, 35]. Recently, a comprehensive study has also been conducted to assess the efficacy of various criteria and provide a fair benchmark [57].

Pruning Grouped Parameters. In complex network structures [29, 32, 38, 71, 77], dependencies can arise among groups of parameters, necessitating their simultaneous
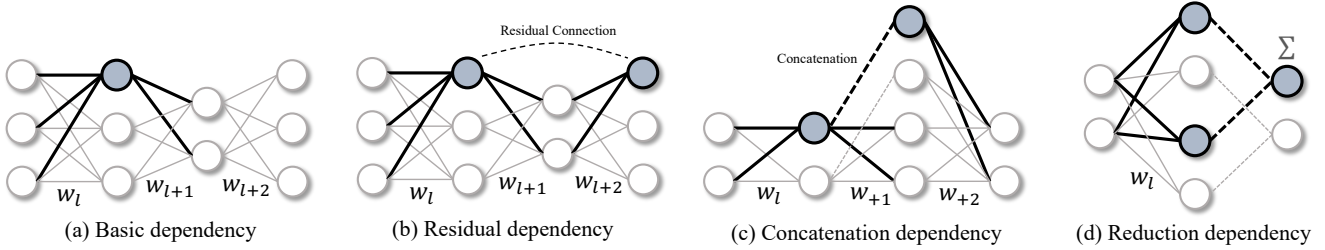
Figure 2. Grouped parameters with inter-dependency in different structures. All highlighted parameters must be pruned simultaneously.

pruning. The pruning of coupled parameters has been a focus of research since the early days of structural pruning [29, 35, 39]. For instance, when pruning two successive convolutional layers, removing a filter from the first layer necessitates the pruning of associated kernels in the subsequent layer [29]. Although it is feasible to analyze parameter dependencies manually, this process can be exceedingly labor-intensive when applied to complicated networks as illustrated in many prior studies [29, 71, 77]. Furthermore, such manual schemes are inherently non-transferable to novel architectures, which severely restricts the applications of pruning. Recently, some pilot works have been proposed to decipher the complex relationships between layers [32, 71]. Unfortunately, existing techniques still depend on empirical rules or predefined architectural patterns, rendering them insufficiently versatile for all structural pruning applications. In this study, we present a general approach to resolve this challenge, demonstrating that addressing parameter dependency effectively generalizes structural pruning across a wide array of networks, resulting in satisfactory performance on several tasks.

3. Method

3.1. Dependency in Neural Networks

In this work, we focus on structural pruning of *any neural networks* under the restriction of parameter dependency. Without loss of generality, we develop our method upon fully-connected (FC) layers. Let's begin with a linear neural network composed of three consecutive layers as illustrated in Figure 2 (a), parameterized by 2-D weight matrices w_l, w_{l+1} and w_{l+2} respectively. This simple neural network can be made slim by structural pruning via the removal of neurons. In this case, it is easy to find that some dependencies emerge between parameters, denoted as $w_l \Leftrightarrow w_{l+1}$, which forces w_l and w_{l+1} to be simultaneously pruned. Specifically, to prune the k-th neuron that bridges w_l and w_{l+1}, both $w_l[k,:]$ and $w_{l+1}[:,k]$ will be removed. In the literature, researchers handle layer dependencies and enable structural pruning on deep neural networks with manually-designed and model-specific schemes [21, 29]. Nevertheless, there are many kinds of dependencies just as illustrated

in Figure 2 (b-d). It is intractable to manually analyze all dependencies in a case-by-case manner, let alone that simple dependencies can be nested or composed to form more complex patterns. To address the dependency issue in structural pruning, we introduce Dependency Graph in this work, which provides a general and fully-automatic mechanism for dependency modeling.

3.2. Dependency Graph

Grouping. To enable structural pruning, we first need to group layers according to their inter-dependency. Formally, our goal is to find a grouping matrix $G \in R^{L \times L}$ where L refers to the number of layers in a to-be-pruned network, and $G_{ij} = 1$ indicates the presence of dependency between i-th layer and j-th layer. We let $\text{Diag}(G) = \mathbf{1}^{1 \times L}$ to enable self-dependency for convenience. With the grouping matrix, it is straightforward to find all coupled layers with inter-dependency to the i-th layer, denoted as $g(i)$:

$$g(i) = \{j | G_{ij} = 1\} \tag{1}$$

Nevertheless, it is non-trivial to estimate the grouping patterns from a neural network due to the fact that modern deep networks may consist of thousands of layers with complicated connections, resulting in a large and intricate grouping matrix G. In this matrix, G_{ij} is not only determined by the i-th and j-th layers but also affected by those intermediate layers between them. Thus, such non-local and implicit relations can not be handled with simple rules in most cases. To overcome this challenge, we do not directly estimate the grouping matrix G, and propose an equivalent but easy-to-estimate method for dependency modeling, namely the Dependency Graph, from which G can be efficiently derived.

Dependency Graph. Let us begin by considering a group $g = \{w_1, w_2, w_3\}$, which has dependencies $w_1 \Leftrightarrow w_2$, $w_2 \Leftrightarrow w_3$, and $w_1 \Leftrightarrow w_3$. Upon closer inspection of this dependency modeling, we can observe that there is some redundancy present. For example, the dependency $w_1 \Leftrightarrow w_3$ can be derived from $w_1 \Leftrightarrow w_2$ and $w_2 \Leftrightarrow w_3$ through a recursive process. Initially, we take w_1 as the starting point and examine its dependency on other layers, such as
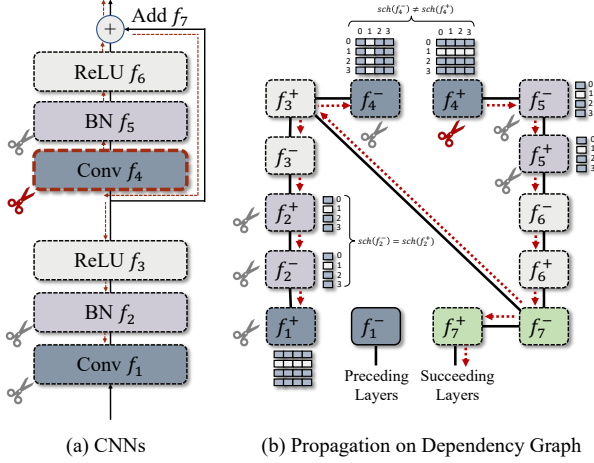
| (a) CNNs | (b) Propagation on Dependency Graph |

Figure 3. Layer grouping is achieved via a recursive propagation on DepGraph, starting from the f_4^+. In this example, there is no Intra-layer Dependency between convolutional input f_4^- and output f_4^+ due to the diverged pruning schemes illustrated above.

$w_1 \Leftrightarrow w_2$. Then, w_2 provides a new starting point for recursively expanding the dependency, which in turn "triggers" $w_2 \Leftrightarrow w_3$. This recursive process ultimately ends with a transitive relation, $w_1 \Leftrightarrow w_2 \Leftrightarrow w_3$. In this case, we only need two dependencies to describe the relations in group g. Similarly, the grouping matrix discussed in Section 3.2 is also redundant for dependency modeling and thus can be compressed into a more compact form with fewer edges while retaining the same information. We demonstrate that a new graph D that measures the local inter-dependency between adjacent layers, named Dependency Graph, can be an effective reduction for the grouping matrix G. The Dependency Graph differs from G in that it only records the dependencies between adjacent layers with direct connections. The Graph D can be viewed as the transitive reduction [1] of G, which contains the same vertices as G but with as few edges as possible. Formally, D is constructed such that, for all $G_{ij} = 1$, there exists a path in D between vertex i and j. Therefore, G_{ij} can be derived by examing the presence of a path between vertices i and j in D.

Network Decomposition. However, we find that building the dependency graph at the layer level can be problematic in practice, since some basic layers such as fully-connected layers may have two different pruning schemes like $w[k, :]$ and $w[:, k]$ as discussed in Section 3.1, which compress the dimensions of inputs and outputs respectively. Besides, networks also contain non-parameterized operations such as skip connections, which also affect the dependency between layers [40]. To remedy these issues, we propose a new notation to decompose a network $\mathcal{F}(x; w)$ into finer and more basic components, denoted as $\mathcal{F} = \{f_1, f_2, ..., f_L\}$, where each component f refers to either a parameterized

layer such as convolution or a non-parameterized operation such as residual adding. Instead of modeling relationships at the layer level, we concentrate on the dependencies between the inputs and outputs of the layers. Specifically, we denote the input and output of component f_i as f^-i and f^+i, respectively. For any network, the final decomposition can be formalized as $\mathcal{F} = \{f_1^-, f_1^+, ..., f_L^-, f_L^+\}$. This notation facilitates easier dependency modeling and allows different pruning schemes for the same layer.

Dependency Modeling. Leveraging this notation, we re-sketch the neural network as Equation 2, where two principal types of dependencies can be discerned, namely inter-layer dependency and intra-layer dependency, as demonstrated below:

$$\underbrace{(f_1^-, f_1^+)}_{} \leftrightarrow (f_2^-, f_2^+) \cdots \leftrightarrow \underbrace{(f_L^-, f_L^+)}_{\text{Intra-layer Dep}} \quad (2)$$
$$\text{Inter-layer Dep}$$

The symbol $\leftrightarrow$ signifies the connectivity between two adjacent layers. Examination of these two dependencies yields straightforward but general rules for dependency modeling:

- *Inter-layer Dependency*: A dependency $f_i^- \Leftrightarrow f_j^+$ consistently arises in connected layers where $f_i^- \leftrightarrow f_j^+$.

- *Intra-layer Dependency*: A dependency $f_i^- \Leftrightarrow f_i^+$ exists if f_i^- and f_i^+ share identical pruning schemes, denoted by $sch(f_i^-) = sch(f_i^+)$.

First, the inter-layer dependency can be readily estimated if the topological structure of the network is known. For connected layers with $f_i^- \leftrightarrow f_j^+$, a dependency consistently exists, because f_i^- and f_j^+ correspond to the same intermediate features of the network. The subsequent step involves elucidating the intra-layer dependency. An intra-layer dependency necessitates that both the input and output of a single layer should be pruned simultaneously. Numerous network layers satisfy this condition, such as batch normalization, whose inputs and outputs share the same pruning scheme, denoted as $sch(f_i^-) = sch(f_i^+)$, and thus will be pruned simultaneously as illustrated in Figure 3. In contrast, layers like convolutions have distinct pruning schemes for their inputs and outputs, i.e., $w[:, k, :, :] \neq w[k, :, :, :]$ as shown in Figure 3, resulting in $sch(f_i^-) \neq sch(f_i^+)$. In such instances, there is no dependency between the input and output of a convolution layer. Given the aforementioned rules, we can formally establish the dependency modeling as follows:

$$D(f_i^-, f_j^+) = \underbrace{\mathbb{1}\left[f_i^- \leftrightarrow f_j^+\right]}_{\text{Inter-layer Dep}} \vee \underbrace{\mathbb{1}\left[i = j \wedge sch(f_i^-) = sch(f_j^+)\right]}_{\text{Intra-layer Dep}}$$
$$(3)$$

where $\vee$ and $\wedge$ refer to the logical "OR" and "AND" operations, and $\mathbb{1}$ is a indicator function returning "True" is

(a) Unstructural Sparsity   **(b) Structural but Inconsistent Sparsity**   **(c) Consistent Structural Sparsity**
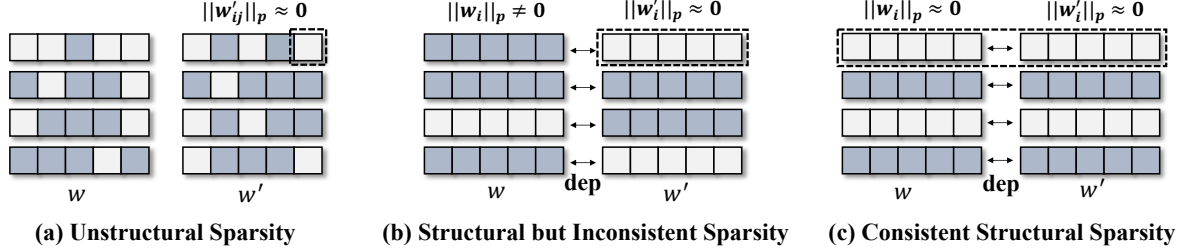
Figure 4. Learning different sparsity schemes to estimate the importance of grouped parameters. Method (a) is used in unstructural pruning which only focuses on the importance of single weight. Method (b) learns structurally sparse layers [35], but ignores coupled weights in other layers. Our method as shown in (c) learns group sparsity which forces all coupled parameters to zero, so that they can be easily distinguished by a simple magnitude method.

Algorithm 1: Dependency Graph

Input: A neural network $\mathcal{F}(x; w)$
Output: DepGraph $D(\mathcal{F}, E)$
$f^- = \{f_1^-, f_2^-, ..., f_L^-\}$ decomposed from the $\mathcal{F}$
$f^+ = \{f_1^+, f_2^+, ..., f_L^+\}$ decomposed from the $\mathcal{F}$
Initialize DepGraph $D = \mathbf{0}_{2L \times 2L}$
for $i = \{0, 1, .., L\}$ **do**
  for $j = \{0, 1, .., L\}$ **do**
    $D(f_i^-, f_j^+) = D(f_j^+, f_i^-) =$
    $\underbrace{\mathbb{1}\left[f_i^- \leftrightarrow f_j^+\right]}_{\text{Inter-layer Dep}} \vee \underbrace{\mathbb{1}\left[i = j \wedge sch(f_i^-) = sch(f_j^+)\right]}_{\text{Intra-layer Dep}}$
return D

Algorithm 2: Grouping

Input: DepGraph $D(\mathcal{F}, E)$
Output: Groups G
$G = \{\}$
for $i = \{1, 2, ..., 2 * \|\mathcal{F}\|\}$ **do**
  $g = \{i\}$
  repeat
    UNSEEN $= \{1, 2, ..., 2 * \|\mathcal{F}\|\} - g$
    $g' = \{j \in \text{UNSEEN} | \exists k \in g, D_{kj} = 1\}$
    $g = g \cup g'$
  until $g' = \emptyset$;
  $G = G \cup \{g\}$
return G

---

the condition holds. The first term examines the Inter-layer Dependency caused by network connectivity, while the second term examines the intra-layer dependency introduced by shared pruning schemes between layer inputs and outputs. It is worth noting that, DepGraph is a symmetric matrix with $D(f_i^-, f_j^+) = D(f_j^+, f_i^-)$. As such, we can examine all input-output pairs to estimate the dependency graph. In Figure 3, we visualize the DepGraph of a CNN block with residual connections. Algorithm 1 and 2 summarize the algorithms for dependency modeling and grouping.

3.3. Group-level Pruning

In previous sections, we have developed a general methodology for analyzing dependencies within neural networks, which naturally leads to a group-level pruning problem. Assessing the importance of grouped parameters poses a significant challenge to pruning as it involves several coupled layers. In this section, we leverage a simple norm-based criterion [29] to establish a practical method for group-level pruning. Given a parameter group $g = \{w_1, w_2, ..., w_{|g|}\}$, existing criteria like L_2-norm importance $I(w) = \|w\|_2$ can produce independent scores for each $w \in g$. A natural way to estimate the group importance would be computing an aggregated score $I(g) = \sum_{w \in g} I(w)$. Unfortunately, importance scores indepen-

dently estimated on different layers are likely to be non-additive and thus meaningless due to the divergence of distributions and magnitudes. To make this simple aggregation works for importance estimation, we propose a sparse training method to sparsify parameters at the group level as illustrated in Figure 4 (c), so that those zeroized groups can be safely removed from the network. Specifically, for each parameters w with K prunable dimensions indexed by $w[k]$, we introduce a simple regularization term for sparse training, defined as:

$$\mathcal{R}(g, k) = \sum_{k=1}^{K} \gamma_k \cdot I_{g,k} = \sum_{k=1}^{K} \sum_{w \in g} \gamma_k \|w[k]\|_2^2 \quad (4)$$

where $I_{g,k} = \sum_{w \in g} \|w[k]\|_2^2$ represents the importance of the k-th prunable dimensions, and γ_k refers to the shrinkage strength applied to those parameters. We use a controllable exponential strategy to determine the γ_k as follows:

$$\gamma_k = 2^{\alpha(I_g^{\max} - I_{g,k})/(I_g^{\max} - I_g^{\min})} \quad (5)$$

where a normalized score is used to control the shrinkage strength α_k, varying within the range of $[2^0, 2^\alpha]$. In this work, we use a constant hyper-parameter $\alpha = 4$ for all experiments. After sparse training, we further use a simple

Model / Data	Method	Base	Pruned	Δ Acc.	Speed Up
ResNet56 CIFAR10	NISP [74]	-	-	-0.03	1.76×
	Geometric [20]	93.59	93.26	-0.33	1.70×
	Polar [78]	93.80	93.83	+0.03	1.88×
	CP [29]	92.80	91.80	-1.00	2.00×
	AMC [19]	92.80	91.90	-0.90	2.00×
	HRank [31]	93.26	92.17	-0.09	2.00×
	SFP [18]	93.59	93.36	-0.23	2.11×
	ResRep [7]	93.71	93.71	+0.00	**2.12×**
	Ours w/o SL	93.53	93.46	-0.07	2.11×
	Ours	93.53	**93.77**	**+0.24**	2.11×
	GBN ([71])	93.10	92.77	-0.33	2.51×
	AFP ([6])	93.93	92.94	-0.99	2.56×
	C-SGD ([4])	93.39	93.44	+0.05	2.55×
	GReg-1 ([58])	93.36	93.18	-0.18	2.55×
	GReg-2 ([58])	93.36	93.36	-0.00	2.55×
	Ours w/o SL	93.53	93.36	-0.17	2.51×
	Ours	93.53	**93.64**	**+0.11**	**2.57×**
VGG19 CIFAR100	OBD ([56])	73.34	60.70	-12.64	5.73×
	OBD ([56])	73.34	60.66	-12.68	6.09×
	EigenD ([56])	73.34	65.18	-8.16	8.80×
	GReg-1 ([58])	74.02	67.55	-6.67	8.84×
	GReg-2 ([58])	74.02	67.75	-6.27	8.84×
	Ours w/o SL	73.50	67.60	-5.44	8.87×
	Ours	73.50	**70.39**	**-3.11**	**8.92×**

Table 1. Pruning results on CIFAR-10 and CIFAR-100.

relative score $\hat{I}_{g,k} = N \cdot I_{g,k} / \sum \{\text{TopN}(I_g)\}$ to identify and remove unimportant parameters. In the experiments section, we show that such a simple pruning method, when combined with consistent sparse training, can achieve comparable performance to modern approaches.

4. Experiments

4.1. Settings

This paper focuses on classification tasks and conducts extensive experiments on a variety of datasets, such as CIFAR [26] and ImageNet [3] for image classification, PPI [14] for graph classification, ModelNet [62] for 3D classification, and AGNews [76] for text classification. For each dataset, we evaluated our method on several popular architectures, including ResNe(X)t [40, 63], VGG [51], DenseNet [23], MobileNet [48], GoogleNet [54], Vision Transformers [10], LSTM [12], DGCNNs [60], and Graph Attention Networks [55]. To conduct ImageNet experiments, we use off-the-shelf models from Torchvision [41] as the original models. After pruning, All models will be fine-tuned following a similar protocol as the pre-training stage, with a smaller learning rate and fewer iterations.

4.2. Results on CIFAR

Performance. CIFAR [26] is a tiny image dataset, which is widely used to verify the effectiveness of pruning algorithms. We follow existing works [7, 58] to prune a ResNet-56 on CIFAR-10, and a VGG network on CIFAR-100. As shown in Table 1. We report the accuracy of the pruned models as well as their theoretical speedup ratios,
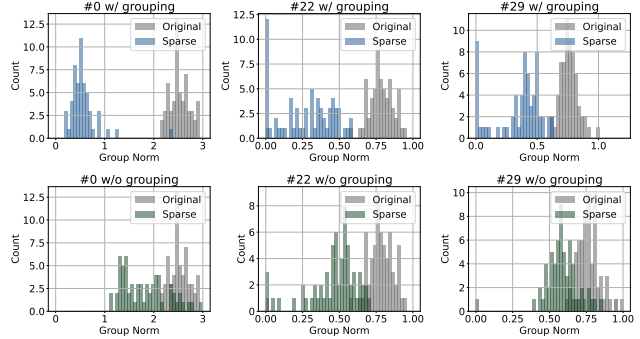

Figure 5. Histogram of group-level sparsity obtained by sparse learning w/ and w/o grouping, which respectively correspond to the strategy (c) and (b) in Figure.4.

defined as Speed Up $= \frac{\text{MACs(base)}}{\text{MACs(pruned)}}$. Note that baselines like ResRep [7], GReg [58] also deploy sparse training for pruning. A key difference between our algorithm and existing sparsity-based algorithms is that our pruner consistently promotes sparsity across all grouped layers, convering convolutions, batch normalizations and fully-connected layers. With this improvement, we are able to take full advantage of the group structures to learn better sparsity, and thus improve the accuracy of pruned models.

Distribution of Group Sparsity. As previously stated, consistent sparsity are important for structural pruning, as it forces all pruned parameters to be consistently unimportant. In Figure 5, we visualize the norm of grouped parameters learned by the consistent and inconsistent strategies in Figure 4 (c) and (b). It is easy to find that our method produces strong sparsity at the group level, which would be beneficial for identifying unimportant parameters. However, the inconsistent method that works on different layers independently fails to produce consistent importance across layers, which could result in non-sparse norm at the group level.

4.3. Ablation Study

Grouping Strategy. To further verify the effectiveness of grouping, we evaluate different strategies on several convolutional networks. Strategies mainly include: 1) No grouping: sparse learning and importance evaluation are performed independently on a single convolutional layer; 2) Conv-only Grouping: all convolutional layers within the group are sparsified in a consistent manner. 3) Full Grouping: All parameterized layers within a group, such as convolutions, batch normalizations, and fully-connected layers, are sparsified consistently. As shown in Table 2, When we ignore the grouping information in neural networks and sparsify each layer in isolation, the performance of our approach will degrade significantly, and in some cases even collapse due to over-pruning. The results of Conv-only set-

Architecture	Strategy	Pruned Accuracy with Uniform / Learned Sparsity				
		1.5×	3.0×	6.0×	12×	Avg.
ResNet-56 (72.58)	Random	71.49 / 72.07	68.52 / 68.16	60.35 / 60.25	53.21 / 48.01	63.39 / 62.15
	No grouping	71.96 / 72.07	67.85 / 67.89	62.64 / 63.18	54.52 / 53.65	64.24 / 64.20
	Conv-only	71.64 / 71.94	68.30 / 69.07	62.44 / 62.63	53.89 / 54.94	64.07 / 64.65
	Full Grouping	71.68 / **72.57**	68.70 / **70.38**	63.72 / **65.33**	55.23 / **55.92**	64.83 / **66.09**
VGG-19 (73.50)	Random	72.63 / 72.77	71.27 / 70.83	68.97 / 69.16	62.45 / 63.42	63.83 / 69.05
	No Grouping	73.83 / 55.13	71.40 / 53.21	69.19 / 50.10	65.12 / [†]3.87	69.14 / 40.58
	Conv-Only	73.32 / 73.22	71.38 / 71.80	69.66 / 69.85	64.69 / 65.95	69.76 / 70.21
	Full Grouping	73.11 / **74.00**	71.57 / **72.46**	69.72 / **70.38**	65.74 / **66.20**	70.03 / **70.58**
DenseNet-121 (78.73)	Random	79.04 / 79.43	77.86 / 78.62	75.47 / 74.52	69.26 / 69.64	75.41 / 75.80
	No Grouping	79.31 / 78.91	78.08 / 78.62	**78.62** / 68.57	72.93 / 57.17	77.24 / 70.82
	Conv-Only	79.18 / **79.74**	77.98 / 78.85	76.61 / 77.22	73.30 / 73.95	76.77 / 77.44
	Full Grouping	79.34 / 79.74	77.97 / **79.19**	77.08 / 77.78	**74.77** / 75.29	77.29 / **77.77**
MobileNetv2 (70.80)	Random	70.90 / 70.69	67.75 / 67.54	61.32 / 62.26	53.41 / 53.97	63.35 / 63.62
	No Grouping	71.16 / 71.28	69.93 / 68.59	66.76 / 37.38	60.28 / 28.24	67.03 / 51.37
	Conv-Only	71.22 / 71.51	70.33 / 70.15	66.16 / 66.49	61.35 / 63.24	67.27 / 67.85
	Full Grouping	71.11 / **71.67**	70.06 / **70.81**	66.48 / **68.02**	60.32 / **63.37**	66.99 / **68.67**
GoogleNet (77.56)	Random	77.52 / 77.72	76.47 / 76.15	74.92 / 74.19	69.37 / 69.69	74.57 / 74.44
	No Grouping	77.44 / 77.23	76.84 / 74.95	75.60 / 63.78	71.92 / 63.72	75.45 / 69.92
	Conv-Only	77.33 / **77.62**	76.68 / 76.92	75.66 / 74.98	71.90 / 71.87	75.49 / 75.35
	Full Grouping	77.91 / 77.76	76.90 / **77.00**	75.42 / **75.44**	71.98 / **72.88**	75.53 / **75.57**

Table 2. Ablation study on CIFAR-100 for different grouping strategies and sparsity configurations. The proposed strategy, full grouping, takes all parameterized layers into account during sparse training, while other strategies only leverage partial layers. Accuracy (%) of pruned models with uniform layer sparsity or learned layer sparsity is reported. †:In some cases, our method over-prunes some dimension to 1, which severely damages the final accuracy.

ting show that grouping partial parameters is beneficial to the final performance, but some useful information in the group is still ignored. Therefore, it is feasible to further improve the pruning accuracy with the full grouping strategy.

Learned Sparsity. Layer sparsity is also an important factor for pruning, which determines the final structure of pruned neural networks. Table 2 provides some results about layer sparsity. This work primarily focuses on two types of sparsity, namely uniform sparsity and learned sparsity. With uniform sparsity, the same pruning ratio will be applied to different layers, assuming that redundancy is distributed uniformly through the network. However, previous experiments in Figure 5 have shown that different layers are not equally prunable. In most cases, the learned sparsity outperforms the uniform one, although sometimes it may over-prune some layers, leading to degraded accuracy.

Generalizability of DepGraph. Results in Table 2 also demonstrate the generalizability of our framework, which is able to handle various convolutional neural networks. Moreover, we emphasize that our method is compatible with DenseNet and GoogleNet, which contains dense connections and parallel structures. In the following sections,

we will further demonstrate the capability of our framework to more architectures.

4.4. Towards Any Structural Pruning

Visualization of DepGraph. Pruning large neural networks presents a considerable challenge due to the intricate process of grouping parameters. However, by employing the DepGraph, all coupled groups can be effortlessly obtained. We provide visualizations of the DepGraph D and the derived grouping matrices G for DenseNet-121 [23], ResNet-18, and Vision Transformers [10] in Figure 6. The grouping matrices are derived from the DepGraph as outlined in Algorithm 2, where $G[i, j] = 1$ signifies that the i-th layer belongs to the same group as the j-th layer. DenseNet-121 demonstrates a strong correlation between layers within the same dense block, leading to large groups during the structural pruning. The proposed Dependency Graph proves to be helpful when dealing with complex networks, as manually analyzing all dependencies in such networks is indeed an intractable task.

ImageNet. Table 3 presents pruning results on ImageNet for several architectures, including ResNet, DenseNet, MobileNet, ResNeXt, and Vision Transformers. The target of

Arch.	Method	Base	Pruned	Δ Acc.	MACs
ResNet-50	ResNet-50	76.15	-	-	4.13
	ThiNet [39]	72.88	72.04	-0.84	2.44
	SSS [24]	76.12	74.18	-1.94	2.82
	SFP [18]	76.15	74.61	-1.54	2.40
	AutoSlim [73]	76.10	75.60	-0.50	2.00
	FPGM [20]	76.15	75.50	-0.65	2.38
	Taylor [42]	76.18	74.50	-1.68	2.25
	Slimable [72]	76.10	74.90	-1.20	2.30
	CCP [45]	76.15	75.50	-0.65	2.11
	AOFP-C1 [5]	75.34	75.63	+0.29	2.58
	TAS [9]	77.46	76.20	-1.26	2.31
	GFP [32]	76.79	76.42	-0.37	2.04
	GReg-2 [58]	76.13	75.36	-0.77	2.77
	Ours	76.15	75.83	-0.32	1.99
DenseNet-121	DenseNet-121	74.44	-	-	2.86
	PSP-1.38G [50]	74.35	74.05	-0.30	1.38
	PSP-0.58G [50]	74.35	70.34	-4.01	0.58
	Ours-1.38G	74.44	73.98	-0.46	1.37
	Ours-0.58G	74.44	70.13	-4.31	0.57
Mob-v2	Mob-v2	71.87	-	-	0.33
	NetAdapt [64]	-	70.00	-	0.24
	Meta [36]	74.70	68.20	-6.50	0.14
	GFP [32]	75.74	69.16	-6.58	0.15
	Ours	71.87	68.46	-3.41	0.15
NeXt-50	ResNeXt-50	77.62	-	-	4.27
	SSS [24]	77.57	74.98	-2.59	2.43
	GFP [32]	77.97	77.53	-0.44	2.11
	Ours	77.62	76.48	-1.14	2.09
ViT-B/16	VIT-B/16	81.07	-	-	17.6
	CP-ViT [52]	77.91	77.36	-0.55	11.7
	Ours+EMA	81.07	79.58	-1.39	10.4
	Ours	81.07	79.17	-1.90	10.4

Table 3. Pruning results on ImageNet.

Arch. & Data	Method	Base	Pruned	Δ	Speedup
LSTM (AGNews)	DepGraph+Random	92.10	91.23	-0.87	16.28×
	DepGraph+CP [29]	92.10	91.50	-0.60	16.28×
	Ours w/o SL	92.10	91.53	-0.57	16.28×
	Ours	92.10	**91.75**	**-0.35**	16.28×
DGCNN (ModelNet40)	DepGraph+Random	92.10	91.05	-1.05	10.05×
	DepGraph+CP [29]	92.10	91.00	-1.10	10.05×
	DepGraph+Slim [35]	92.10	91.74	-0.36	10.35×
	Ours w/o SL	92.10	91.86	-0.24	11.46×
	Ours	92.10	**92.02**	**-0.08**	11.98×
GAT (PPI)	DepGraph+Random	0.986	0.951	-0.035	8.05×
	DepGraph+CP [29]	0.986	0.957	-0.029	8.05×
	Ours w/o SL	0.986	0.953	-0.033	8.26×
	Ours	0.986	**0.961**	**-0.025**	8.43×

Table 4. Pruning neural networks for non-image data, including AGNews (text), ModelNet (3D Point Cloud) and PPI (Graph). We report the classification accuracy (%) of pruned model for AG-News and ModelNet and micro-F1 score for PPI.
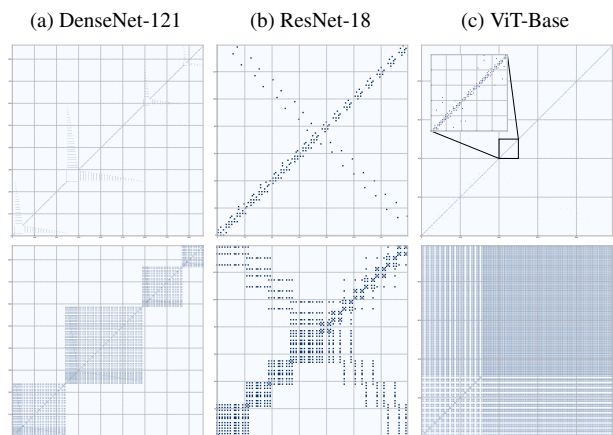
(a) DenseNet-121	(b) ResNet-18	(c) ViT-Base

Figure 6. Dependency graphs (top) and the derived grouping schemes (bottom) for DenseNet-121, ResNet-18 and ViT-Base.

this work is not to provide state-of-the-art results for various models, thus we only use the most basic importance criterion in this work. We show that a simple norm-based criterion, when combined with dependency modeling, can achieve comparable performance to modern approaches that use powerful criteria [32,71] and training techniques [58].

Text, 3D Point Cloud, Graph and More. In addition to CNNs and Transformers, our method is easily applicable to other architectures as well. This part consists of experiments on a variety of data, including texts, graphs, and 3D point clouds, as shown in Table 4. We utilize LSTM for text classification by studying the effectiveness of DepGraph on recursive structures in which parameterized layers are coupled due to the element-wise operations. DepGraph is also tested on Dynamic Graph CNNs that contain aggregation operations for 3D point clouds. Furthermore, we conduct experiments with graph data, which require entirely different architectures from those used for other tasks. In this ex-

periment, we concentrate on the acceleration of Graph Attention Networks, which have several coupled layers within each GNN layer. Considering the lack of works concerning pruning on these datasets, we combine DepGraph with some classic pruning methods in CNNs to establish our baselines. The results indicate that our method can be indeed generalized to a wide variety of architectures.

5. Conclusion

In this work, we introduce Dependency Graph to enable *any structural pruning* on a wide variety of neural networks. Our work is the first attempt, to our knowledge, to develop a general algorithm that can be applied to architectures, including CNNs, RNNs, GNNs, and Transformers.

Acknowledgment

References

[1] Alfred V. Aho, Michael R Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972. 4

[2] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1518–1528, 2020. 2

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6

[4] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4943–4953, 2019. 1, 2, 6

[5] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. In *International Conference on Machine Learning*, pages 1607–1616. PMLR, 2019. 8

[6] Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 6

[7] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4510–4520, 2021. 1, 2, 6

[8] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in Neural Information Processing Systems*, 30, 2017. 1, 2

[9] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. *Advances in Neural Information Processing Systems*, 32, 2019. 8

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 2, 6, 7

[11] Shangqian Gao, Feihu Huang, Weidong Cai, and Heng Huang. Network pruning via performance maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9270–9280, 2021. 1

[12] Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012. 2, 6

[13] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *Advances in neural information processing systems*, 29, 2016. 1

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 6

[15] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 2

[16] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015. 1

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[18] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018. 6, 8

[19] 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 *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800, 2018. 2, 6

[20] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4340–4349, 2019. 2, 6, 8

[21] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017. 2, 3

[22] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 1

[23] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 1, 2, 6, 7

[24] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018. 8

[25] Yongcheng Jing, Yiding Yang, Xinchao Wang, Mingli Song, and Dacheng Tao. Meta-aggregator: Learning to aggregate for 1-bit graph neural networks. In *ICCV*, 2021. 1

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[27] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning. *arXiv preprint arXiv:2010.07611*, 2020. 2

[28] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip HS Torr. A signal propagation perspective for pruning neural networks at initialization. *arXiv preprint arXiv:1906.06307*, 2019. 2

[29] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 1, 2, 3, 5, 6, 8

[30] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021. 1

[31] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1529–1538, 2020. 1, 6

[32] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021. 2, 3, 8

[33] Songhua Liu, Kai Wang, Xingyi Yang, Jingwen Ye, and Xinchao Wang. Dataset distillation via factorization. In *Conference on Neural Information Processing Systems*, 2022. 1

[34] Songhua Liu, Jingwen Ye, Runpeng Yu, and Xinchao Wang. Slimmable dataset condensation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. 1

[35] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017. 2, 3, 5, 8

[36] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3296–3305, 2019. 8

[37] Ekdeep Singh Lubana and Robert P Dick. A gradient flow framework for analyzing network pruning. *arXiv preprint arXiv:2009.11839*, 2020. 2

[38] Jian-Hao Luo and Jianxin Wu. Neural network pruning with residual-connections and limited-data. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1458–1467, June 2020. 1, 2

[39] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017. 2, 3, 8

[40] Xiaolong Ma, Geng Yuan, Sheng Lin, Zhengang Li, Hao Sun, and Yanzhi Wang. Resnet can be pruned 60×: Introducing network purification and unused path removal (p-rm) after weight pruning. In *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 1–2. IEEE, 2019. 2, 4, 6

[41] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1485–1488, 2010. 6

[42] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019. 2, 8

[43] Laurent Orseau, Marcus Hutter, and Omar Rivasplata. Logarithmic pruning is all you need. *Advances in Neural Information Processing Systems*, 33:2925–2934, 2020. 2

[44] Sejun Park, Jaeho Lee, Sangwoo Mo, and Jinwoo Shin. Lookahead: a far-sighted alternative of magnitude-based pruning. *arXiv preprint arXiv:2002.04809*, 2020. 1, 2

[45] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. Collaborative channel pruning for deep networks. In *International Conference on Machine Learning*, pages 5113–5122. PMLR, 2019. 8

[46] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10428–10436, 2020. 2

[47] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020. 2

[48] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 2, 6

[49] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389, 2020. 2

[50] Günther Schindler, Wolfgang Roth, Franz Pernkopf, and Holger Fröning. Parameterized structured pruning for deep neural networks. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 16–27. Springer, 2020. 8

[51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2, 6

[52] Zhuoran Song, Yihong Xu, Zhezhi He, Li Jiang, Naifeng Jing, and Xiaoyao Liang. Cp-vit: Cascade vision transformer pruning via progressive sparsity prediction. *arXiv preprint arXiv:2203.04570*, 2022. 8

[53] Ralf C Staudemeyer and Eric Rothstein Morris. Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019. 2

[54] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 2, 6

[55] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 2, 6

[56] Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *International Conference on Machine Learning*, pages 6566–6575. PMLR, 2019. 6

[57] Huan Wang, Can Qin, Yue Bai, and Yun Fu. Why is the state of neural network pruning so confusing? on the fairness, comparison setup, and trainability in network pruning. *arXiv preprint arXiv:2301.05219*, 2023. 2

[58] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. *arXiv preprint arXiv:2012.09243*, 2020. 1, 2, 6, 8

[59] 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 *International Conference on Machine Learning*, pages 10717–10726. PMLR, 2021. 1

[60] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019. 2, 6

[61] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4820–4828, 2016. 1

[62] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 6

[63] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 2, 6

[64] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018. 8

[65] Xingyi Yang, Jingwen Ye, and Xinchao Wang. Factorizing knowledge in neural networks. In *European Conference on Computer Vision*, 2022. 1

[66] Xingyi Yang, Daquan Zhou, Songhua Liu, Jingwen Ye, and Xinchao Wang. Deep model reassembly. In *Conference on Neural Information Processing Systems*, 2022. 1

[67] Yiding Yang, Jiayan Qiu, Mingli Song, Dacheng Tao, and Xinchao Wang. Distilling knowledge from graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 1

[68] Lewei Yao, Renjie Pi, Hang Xu, Wei Zhang, Zhenguo Li, and Tong Zhang. Joint-detnas: upgrade your detector with nas, pruning and dynamic distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10175–10184, 2021. 1

[69] Jingwen Ye, Songhua Liu, and Xinchao Wang. Partial network cloning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. 1

[70] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018. 2

[71] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *Advances in neural information processing systems*, 32, 2019. 1, 2, 3, 6, 8

[72] Jiahui Yu. Slimmable neural networks for edge devices. 2019. 8

[73] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019. 8

[74] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9194–9203, 2018. 1, 6

[75] Ruonan Yu, Songhua Liu, and Xinchao Wang. Dataset distillation: A comprehensive review. *arXiv preprint arXiv:2301.07014*, 2023. 1

[76] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015. 6

[77] Yulun Zhang, Huan Wang, Can Qin, and Yun Fu. Aligned structured sparsity learning for efficient image super-resolution. *Advances in Neural Information Processing Systems*, 34:2695–2706, 2021. 2, 3

[78] Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. *Advances in neural information processing systems*, 33:9865–9877, 2020. 6