Dep-L_0: Improving L_0-based Network Sparsification via Dependency Modeling

Yang Li, and Shihao Ji ✉

Georgia State University, GA, USA
yli93@student.gsu.edu, sji@gsu.edu

Abstract. Training deep neural networks with an L_0 regularization is one of the prominent approaches for network pruning or sparsification. The method prunes the network during training by encouraging weights to become exactly zero. However, recent work of Gale et al. [11] reveals that although this method yields high compression rates on smaller datasets, it performs inconsistently on large-scale learning tasks, such as ResNet50 on ImageNet. We analyze this phenomenon through the lens of variational inference and find that it is likely due to the independent modeling of binary gates, the mean-field approximation [2], which is known in Bayesian statistics for its poor performance due to the crude approximation. To mitigate this deficiency, we propose a dependency modeling of binary gates, which can be modeled effectively as a multi-layer perceptron (MLP). We term our algorithm Dep-L_0 as it prunes networks via a dependency-enabled L_0 regularization. Extensive experiments on CIFAR10, CIFAR100 and ImageNet with VGG16, ResNet50, ResNet56 show that our Dep-L_0 outperforms the original L_0-HC algorithm of Louizos et al. [32] by a significant margin, especially on ImageNet. Compared with the state-of-the-arts network sparsification algorithms, our dependency modeling makes the L_0-based sparsification once again very competitive on large-scale learning tasks. Our source code is available at https://github.com/leo-yangli/dep-l0.

Keywords: Network Sparsification · L_0-norm Regularization · Dependency Modeling

1 Introduction

Convolutional Neural Networks (CNNs) have achieved great success in a broad range of tasks. However, the huge model size and high computational price make the deployment of the state-of-the-art CNNs to resource-limited embedded systems (e.g., smart phones, drones and surveillance cameras) impractical. To alleviate this problem, substantial efforts have been made to compress and speed up the networks [6]. Among these efforts, network pruning has been proved to be an effective way to compress the model and speed up inference without losing noticeable accuracy [24, 13, 38, 32, 26, 40, 9, 36].

The existing network pruning algorithms can be roughly categorized into two categories according to the pruning granularity: unstructured pruning [24, 13, 9, 36] and structured pruning [25, 38, 31, 43, 8, 40, 27]. As shown in Fig. 1, unstructured pruning

includes weight-level, vector-level and kernel-level pruning, while structured pruning normally refers to filter-level pruning. Although unstructured pruning methods usually lead to higher prune rates than structured ones, they require specialized hardware or software to fully utilize the benefits induced by the high prune rates due to the irregular network structures yielded by unstructured pruning. On the other hand, structured pruning can maintain the regularity of network structures, while pruning the networks effectively, and hence can fully utilize the parallel computing resources of general-purpose CPUs or GPUs. Because of this, in recent years structured pruning has attracted a lot of attention and achieved impressive performances [31, 43, 8, 27]. In this work, we focus on structured pruning, more specifically, filter-level pruning.

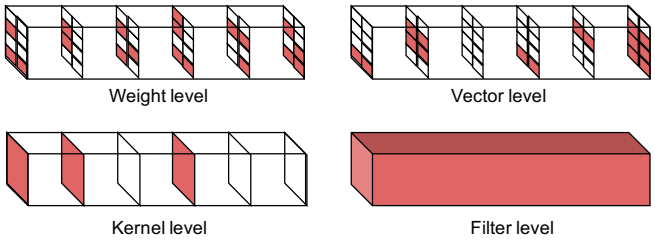


Fig. 1. Visualization of the weights of a convolutional filter and different pruning granularities. The red regions highlight the weights that can be pruned by different pruning methods. This paper focuses on filter-level pruning.

In terms of pruning methods, a simple yet effective strategy is heuristic-based, e.g., pruning the weights based on their magnitudes [24, 13, 25, 19]. Another popular approach is to penalize the model size via sparsity inducing regularization, such as L_1 or L_0 regularization [30, 38, 32]. Among them, L_0-HC [32] is one of the state-of-the-art pruning algorithms that incorporates L_0 regularization for network pruning and has demonstrated impressive performances on many image classification benchmarks (e.g., MNIST, CIFAR10 and CIFAR100). This method attaches a binary gate to each weight of a neural network, and penalizes the complexity of the network, measured by the L_0 norm of the weight matrix. However, recent work of Gale et al. [11] reveals that although L_0-HC works well on smaller datasets, it fails to prune very deep networks on large-scale datasets, such as ResNet50 on ImageNet. The original L_0-HC algorithm was proposed and evaluated on filter-level pruning, while Gale et al. [11] focus on the weight-level pruning. Therefore, it is unclear if the observation of [11] is due to pruning granularity or the deficiency of the L_0 regularization based method. To understand this, we evaluate the original L_0-HC to sparsify ResNet50 at filter level on ImageNet, and find that it indeed cannot prune ResNet50 without a significant damage of model quality, confirming the observation made by [11]. This indicates that the failure of L_0-HC is likely due to the deficiency of the L_0-norm based approach. We further analyze L_0-HC in the lens of variational inference [2], and find that **the failure is likely due to an over-simplified assumption that models the variational posterior of binary gates to be element-wise independent**. To verify this hypothesis, we propose to in-

corporate the dependency into the binary gates, and model the gate dependency across CNN layers with a multi-layer perceptron (MLP). Extensive experiments show that our dependency-enabled L_0 sparsification, termed Dep-L_0, once again is able to prune very deep networks on large-scale datasets, while achieving competitive or sometimes even better performances than the state-of-the-art pruning methods.

Our main contributions can be summarized as follows:

- From a variational inference perspective, we show that the effectiveness of L_0-HC [32] might be hindered by the implicit assumption that all binary gates attached to a neural network are independent to each other. To mitigate this issue, we propose Dep-L_0 that incorporates the dependency into the binary gates to improve the original L_0-based sparsification method.
- A series of experiments on multiple datasets and multiple modern CNN architectures demonstrate that Dep-L_0 improves L_0-HC consistently, and is very competitive or sometimes even outperforms the state-of-the-art pruning algorithms.
- Moreover, Dep-L_0 converges faster than L_0-HC in terms of network structure search, and reduces the time to solution by 20%-40% compared to L_0-HC in our experiments.

2 Related Work

Model compression [6] aims to reduce the size of a model and speed up its inference at the same time. Recently, there has been a flurry of interest in model compression, ranging from network pruning [24, 13, 32, 26, 28], quantization and binarization [4, 12], tensor decomposition [22, 7], and knowledge distillation [18]. Since our algorithm belongs to the category of network pruning, we mainly focus on reviewing related work in pruning.

Network Pruning A large subset of pruning methods is heuristic-based, which assigns an importance score to each weight and prune the weights whose importance scores are below a threshold. The importance scores are usually devised according to types of networks, e.g., the magnitude of weights [24, 13] (Feed-forward NNs), the L_1 or L_2 norm of filters [25] (CNNs), and the average percentage of zero activations [19] (CNNs). However, Ye et al. [39] point out that the assumption that weights/filters of smaller norms are less important may not hold in general, challenging the heuristic-based approaches. These methods usually follow a three-step training procedure: training - pruning - retraining in order to achieve the best performance.

Another subset of pruning methods focuses on training networks with sparsity inducing regularizations. For example, L_2 and L_1 regularizations [30, 38] or L_0 regularization [32] can be incorporated into the objective functions to train sparse networks. Similarly, Molchanov et al. [34] propose variational dropout, a sparse Bayesian learning algorithm under an improper logscale uniform prior, to induce sparsity. In this framework, network pruning can be performed from scratch and gradually fulfilled during training without separated training stages.

Recently, Gale et al. [11] evaluate three popular pruning methods, including variational dropout [34], L_0-HC [32] and magnitude-based pruning [42], on two large-scale

benchmarks. They reveal that although L_0-HC [32] is more sophisticated and yields state-of-the-art results on smaller datasets, it performs inconsistently on large-scale learning task of ImageNet. This observation motivates our development of Dep-L_0. To mitigate the deficiency of L_0-HC, we propose dependency modeling, which makes the L_0-based pruning once again competitive on large-scale learning tasks.

Dependency Modelling Even though there are many network pruning algorithms today, most of them (if not all) *implicitly* assume all the neurons of a network are independent to each other when selecting neurons for pruning. There are quite few works exploring the dependency inside neural networks for pruning. The closest one is LookAhead [36], which reinterprets the magnitude-based pruning as an optimization of the Frobenius distortion of a single layer, and improves the magnitude-based pruning by optimizing the Frobenius distortion of multiple layers, considering previous layer and next layer. Although the interaction of different layers is considered, the authors do not model the dependency of them explicitly. To the best of our knowledge, our Dep-L_0 is the first to model the dependency of neurons explicitly for network pruning.

3 Method

Our algorithm is motivated by L_0-HC [32], which prunes neural networks by optimizing an L_0 regularized loss function and relaxing the non-differentiable Bernoulli distribution with the Hard Concrete (HC) distribution. Since L_0-HC can be viewed as a special case of variational inference under the spike-and-slab prior [32], in this section we first formulate the sparse structure learning from this perspective, then discuss the deficiency of L_0-HC and propose dependency modeling, and finally present Dep-L_0.

3.1 Sparse Structure Learning

Consider a dataset $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$ that consists of N pairs of instances, where $\boldsymbol{x}_i$ is the ith observed data and y_i is the associated class label. We aim to learn a model $p(\mathcal{D}|\boldsymbol{\theta})$, parameterized by $\boldsymbol{\theta}$, which fits $\mathcal{D}$ well with the goal of achieving good generalization to unseen test data. In order to sparsify the model, we introduce a set of binary gates $\boldsymbol{z} = \{z_1, \cdots, z_{|\boldsymbol{\theta}|}\}$, one gate for each parameter, to indicate whether the corresponding parameter being kept ($z = 1$) or not ($z = 0$).

This formulation is closely related to the spike-and-slab distribution [33], which is widely used as a prior in Bayesian inference to impose sparsity. Specifically, the spike-and-slab distribution defines a mixture of a delta spike at zero and a standard Gaussian distribution:

$$\begin{aligned} p(z) &= \text{Bern}(z|\pi) \\ p(\theta|z=0) = \delta(\theta), &\quad p(\theta|z=1) = \mathcal{N}(\theta|0,1), \end{aligned} \tag{1}$$

where $\text{Bern}(\cdot|\pi)$ is the Bernoulli distribution with parameter π, $\delta(\cdot)$ is the Dirac delta function, i.e., a point probability mass centered at the origin, and $\mathcal{N}(\theta|0,1)$ is the Gaussian distribution with zero mean and unit variance. Since both $\boldsymbol{\theta}$ and $\boldsymbol{z}$ are vectors, we assume the prior $p(\boldsymbol{\theta}, \boldsymbol{z})$ factorizes over the dimensionality of $\boldsymbol{z}$.

In Bayesian statistics, we would like to estimate the posterior of $(\boldsymbol{\theta}, \boldsymbol{z})$, which can be calculated by Bayes' rule:

$$p(\boldsymbol{\theta}, \boldsymbol{z}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta}, \boldsymbol{z})p(\boldsymbol{\theta}, \boldsymbol{z})}{p(\mathcal{D})}. \tag{2}$$

Practically, the true posterior distribution $p(\boldsymbol{\theta}, \boldsymbol{z}|\mathcal{D})$ is intractable due to the non-conjugacy of the model likelihood and the prior. Therefore, here we approximate the posterior distribution via variational inference [2]. Specially, we can approximate the true posterior with a parametric variational posterior $q(\boldsymbol{\theta}, \boldsymbol{z})$, the quality of which can be measured by the Kullback-Leibler (KL) divergence:

$$KL[q(\boldsymbol{\theta}, \boldsymbol{z})||p(\boldsymbol{\theta}, \boldsymbol{z}|\mathcal{D})], \tag{3}$$

which is again intractable, but can be optimized by maximizing the variational lower bound of $\log p(\mathcal{D})$, defined as

$$L = \mathbb{E}_{q(\boldsymbol{\theta}, \boldsymbol{z})}[\log p(\mathcal{D}|\boldsymbol{\theta}, \boldsymbol{z})] - KL[q(\boldsymbol{\theta}, \boldsymbol{z})\|p(\boldsymbol{\theta}, \boldsymbol{z})], \tag{4}$$

where the second term can be further expanded as:

$$\begin{aligned} KL[q(\boldsymbol{\theta}, \boldsymbol{z})\|p(\boldsymbol{\theta}, \boldsymbol{z})] =& \mathbb{E}_{q(\boldsymbol{\theta}, \boldsymbol{z})}[\log q(\boldsymbol{\theta}, \boldsymbol{z}) - \log p(\boldsymbol{\theta}, \boldsymbol{z})] \\ =& \mathbb{E}_{q(\boldsymbol{\theta}, \boldsymbol{z})}[\log q(\boldsymbol{\theta}|\boldsymbol{z}) - \log p(\boldsymbol{\theta}|\boldsymbol{z}) + \log q(\boldsymbol{z}) - \log p(\boldsymbol{z})] \\ =& KL[q(\boldsymbol{\theta}|\boldsymbol{z})||p(\boldsymbol{\theta}|\boldsymbol{z})] + KL[q(\boldsymbol{z})||p(\boldsymbol{z})]. \end{aligned} \tag{5}$$

In L_0-HC [32], the variational posterior $q(\boldsymbol{z})$ is factorized over the dimensionality of $\boldsymbol{z}$, i.e., $q(\boldsymbol{z}) = \prod_{j=1}^{|\boldsymbol{\theta}|} q(z_j) = \prod_{j=1}^{|\boldsymbol{\theta}|} \text{Bern}(z_j|\pi_j)$. By the law of total probability, we can further expand Eq. 5 as

$$\begin{aligned} & KL[q(\boldsymbol{\theta}, \boldsymbol{z})\|p(\boldsymbol{\theta}, \boldsymbol{z})] \\ &= \sum_{j=1}^{|\boldsymbol{\theta}|} \Big(q(z_j = 0) KL[q(\theta_j|z_j = 0)||p(\theta_j|z_j = 0)] + \\ &\quad + q(z_j = 1) KL[q(\theta_j|z_j = 1)||p(\theta_j|z_j = 1)] \Big) + \sum_{j=1}^{|\boldsymbol{\theta}|} KL[q(z_j)||p(z_j)] \\ &= \sum_{j=1}^{|\boldsymbol{\theta}|} q(z_j = 1) KL[q(\theta_j|z_j = 1)||p(\theta_j|z_j = 1)] + \sum_{j=1}^{|\boldsymbol{\theta}|} KL[(q(z_j)||p(z_j)]. \end{aligned} \tag{6}$$

The last step holds because $KL[q(\theta_j|z_j = 0)||p(\theta_j|z_j = 0)] = KL[q(\theta_j|z_j = 0)||\delta(\theta_j)] = 0$.

Furthermore, letting $\boldsymbol{\theta} = \tilde{\boldsymbol{\theta}} \odot \boldsymbol{z}$ and assuming $\lambda = KL[q(\theta_j|z_j = 1)||p(\theta_j|z_j = 1)]$, the lower bound L (4) can be simplified as

$$\begin{aligned} L &= \mathbb{E}_{q(\boldsymbol{z})}[\log p(\mathcal{D}|\tilde{\boldsymbol{\theta}} \odot \boldsymbol{z})] - \sum_{j=1}^{|\boldsymbol{\theta}|} KL\left(q\left(z_j\right)||p\left(z_j\right)\right) - \lambda \sum_{j=1}^{|\boldsymbol{\theta}|} q\left(z_j = 1\right) \\ &\leq \mathbb{E}_{q(\boldsymbol{z})}[\log p(\mathcal{D}|\tilde{\boldsymbol{\theta}} \odot \boldsymbol{z})] - \lambda \sum_{j=1}^{|\boldsymbol{\theta}|} \pi_j, \end{aligned} \tag{7}$$

where the inequality holds due to the non-negativity of KL-divergence.

Given that our model is a neural network $h(\boldsymbol{x}; \tilde{\boldsymbol{\theta}}, \boldsymbol{z})$, parameterized by $\tilde{\boldsymbol{\theta}}$ and $\boldsymbol{z}$, Eq. 7 turns out to be an L_0-regularized loss function [32]:

$$\mathcal{R}(\tilde{\boldsymbol{\theta}}, \boldsymbol{\pi}) = \mathbb{E}_{q(\boldsymbol{z})} \left[\frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(h(\boldsymbol{x}_i; \tilde{\boldsymbol{\theta}} \odot \boldsymbol{z}), y_i\right) \right] + \lambda \sum_{j=1}^{|\boldsymbol{\theta}|} \pi_j, \tag{8}$$

where $\mathcal{L}(\cdot)$ is the cross entropy loss for classification.

In the derivations above, the variational posterior $q(\boldsymbol{z})$ is assumed to factorize over the dimensionality of $\boldsymbol{z}$, i.e., $q(\boldsymbol{z}) = \prod_{j=1}^{|\boldsymbol{\theta}|} q(z_j)$. This means all the binary gates $\boldsymbol{z}$ are assumed to be *independent* to each other – the mean-field approximation [2]. In variational inference, it is common to assume the prior $p(\boldsymbol{z})$ to be element-wise independent; the true posterior $p(\boldsymbol{z}|\mathcal{D})$, however, is unlikely to be element-wise independent. Therefore, approximating the true posterior by an element-wise independent $q(\boldsymbol{z})$ is a very restrict constraint that limits the search space of admissible $q(\boldsymbol{z})$ and is known in Bayesian statistics for its poor performance [1, 2]. We thus hypothesize that this mean-field approximation may be the cause of the failure reported by Gale et al. [11], and the independent assumption hinders the effectiveness of the L_0-based pruning method. Therefore, we can potentially improve L_0-HC by relaxing this over-simplified assumption and modeling the dependency among binary gates $\boldsymbol{z}$ explicitly.

Specifically, instead of using a fully factorized variational posterior $q(\boldsymbol{z})$, we can model $q(\boldsymbol{z})$ as a conditional distribution by using the chain rule of probability

$$q(\boldsymbol{z}) = q(z_1)q(z_2|z_1)q(z_3|z_1, z_2) \cdots q(z_{|\theta|}|z_1, \cdots, z_{|\boldsymbol{\theta}|-1}),$$

where given an order of binary gates $\boldsymbol{z} = \{z_1, z_2, \cdots, z_{|\theta|}\}$, z_i is dependent on all previous gates $z_{<i}$. With this, Eq. 8 can be rewritten as

$$\mathcal{R}(\tilde{\boldsymbol{\theta}}, \boldsymbol{\pi}) = \lambda \sum_{j=1}^{|\boldsymbol{\theta}|} \pi_j + \mathbb{E}_{q(z_1) \cdots q(z_{|\boldsymbol{\theta}|}|z_1, \cdots, z_{|\boldsymbol{\theta}|-1})} \left[\frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(h(\boldsymbol{x}_i; \tilde{\boldsymbol{\theta}} \odot \boldsymbol{z}), y_i\right) \right], \tag{9}$$

which is a dependency-enabled L_0 regularized loss function for network pruning. Detailed design of the dependency modeling is to be discussed in later sections.

3.2 Group Sparsity

So far we have modeled a sparse network by attaching a set of binary gates $\boldsymbol{z}$ to the network at the weight level. As we discussed in the introduction, we prefer to prune

the network at the filter level to fully utilize general purpose CPUs or GPUs. To this end, we consider group sparsity that shares a gate within a group of weights. Let $G = \{g_1, g_2, \cdots, g_{|G|}\}$ be a set of groups, where each element corresponds to a group of weights, and $|G|$ is the number of groups. With the group sparsity, the expected L_0-norm of model parameters (the first term of Eq. 9) can be calculated as

$$\mathbb{E}_{q(\boldsymbol{z})}\|\boldsymbol{\theta}\|_0 = \sum_{j=1}^{|\boldsymbol{\theta}|} q(z_j = 1|z_{<j}) = \sum_{k=1}^{|G|} |g_k|\pi_k, \tag{10}$$

where $|g_k|$ denotes the number of weights in group k.

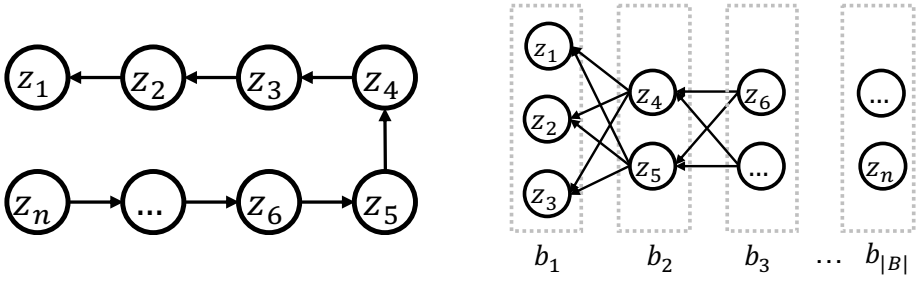
In all our experiments, we perform filter-level pruning by attaching a binary gate to all the weights of a filter (i.e., a group). Since modern CNN architectures often contain batch normalization layers [21], in our implementation we make a slight modification that instead of attaching the gates to filters directly, we attach the gates to the feature maps after batch normalization. This is because batch normalization accumulates a moving average of feature statistics for normalization during the training process. Simply attaching a binary gate to the weights of a filter cannot remove the impact of a filter completely when $z = 0$ due to the memorized statistics from batch normalization. By attaching the gates to the feature maps after batch normalization, the impact of the corresponding filter can be completely removed when $z = 0$.

3.3 Gate Partition

Modern CNN architectures, such as VGGNet [37], ResNet [14] and WideResNet [41], often come with a large number of weights and filters. For example, VGG16 [37] contains 138M parameters and 4,224 filters. Since we attach a binary gate to each filter, the number of gates would be large and modeling the dependencies among them would lead to a huge computational overhead and optimization issues. To make our dependency modeling more practical, we propose gate partition to simplify the dependency modeling among gates. Specifically, the gates are divided into blocks, and the gates within each block are considered independent to each other, whereas the gates cross blocks are considered dependent. Fig. 2 illustrates the difference between an element-wise sequential dependency modeling and a partition-wise dependency modeling. Let's consider z_1, z_2, z_3 and z_4 in these two cases. In the element-wise sequential dependency modeling, as shown in Fig. 2(a), z_2 is dependent on z_1, z_3 is dependent on z_1 and z_2, and so on. As number of gates could be very large, the element-wise sequential modeling would lead to a very long sequence, whose calculation would incur huge computational overhead. Instead, we can partition the gates, as in Fig. 2(b), where z_1, z_2 and z_3 are in block $\boldsymbol{b}_1$ so they are considered independent to each other, while z_4 in block $\boldsymbol{b}_2$ is dependent on all z_1, z_2 and z_3.

We formally describe the gate partition as following. Given a set of gates $G = \{g_1, g_2, \cdots, g_{|G|}\}$, let $B = \{b_1, b_2, \cdots, b_{|B|}\}$ be a partition of G, where b_i denotes block i, and $|B|$ is the total number of blocks. Then we can approximate the variational posterior of $\boldsymbol{z}$ by modeling the distribution over blocks as

$$q(\boldsymbol{z}) \approx q(b_1)q(b_2|b_1)q(b_3|b_1, b_2)\cdots q(b_{|B|}|b_1, \cdots, b_{|B|-1}).$$



(a) element-wise dependency (b) partition-wise dependency

Fig. 2. Illustration of (a) element-wise sequential dependency modeling, and (b) partition-wise dependency modeling.

To reduce the complexity, we can further simplify it as

$$q(\boldsymbol{z}) \approx q(b_1)q(b_2|b_1)q(b_3|b_2)\cdots q(b_{|B|}|b_{|B|-1}), \tag{11}$$

where block i only depends on previous block $i-1$, ignoring all the other previous blocks, i.e., $q(b_i|b_{i-1})$, – the first-order Markov assumption.

In our experiments, we define a layer-wise partition, i.e., a block containing all the filters in one layer. For example, in VGG16 after performing a layer-wise gate partition, we only need to model the dependency within 16 blocks instead of 4,224 gates, and therefore the computational overhead can be reduced significantly.

3.4 Neural Dependency Modeling

Until now we have discussed the dependency modeling in a mathematical form. To incorporate the dependency modeling into the original deep network, we adopt neural networks to model the dependencies among gates. Specifically, we choose to use an MLP network as the *gate generator*. With the proposed *layer-wise* gate partition (i.e., attaching an MLP layer to a convolutional layer and a gate to a filter; see Fig. 2(b)), the MLP architecture can model the dependency of gates, as expressed in Eq. 11, effectively.

Formally, we represent the gate generator as an MLP, with gen_l denoting the operation of the lth layer. The binary gate z_{lk} (i.e. the kth gate in block l) can be generated by

$$\begin{aligned} &\log \alpha_0 = \mathbf{1} \\ &\log \alpha_l = gen_l(\log \alpha_{l-1}) \quad \text{with } gen_l(\cdot) = c \cdot \tanh(W_l \cdot), \\ &z_{lk} \sim \text{HC}(\log \alpha_{lk}, \beta), \end{aligned} \tag{12}$$

where W_l is the weight matrix of MLP at the lth layer, c is a hyperparameter that bounds the value of $\log \alpha$ in the range of $(-c, c)$, and $\text{HC}(\log \alpha, \beta)$ is the Hard Concrete distribution with the location parameter $\log \alpha$ and the temperature parameter β [32][1],

[1] Following L_0-HC [32], β is fixed to 2/3 in our experiments.

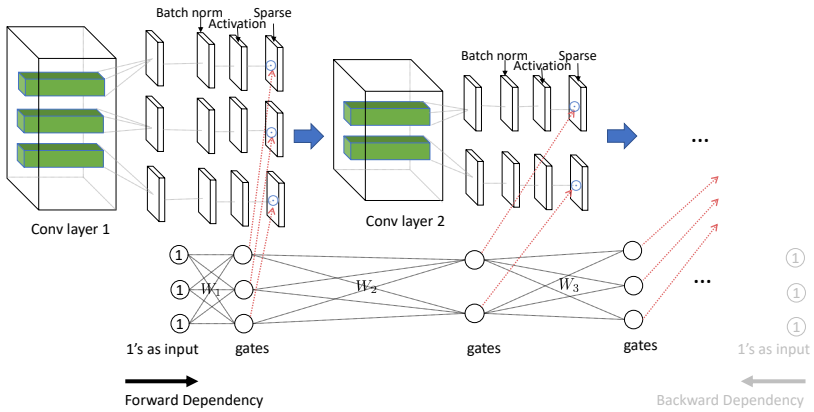


Fig. 3. The computational graph of Dep-L_0. The original CNN network is shown on the top, and the gate generator network (MLP) is shown at the bottom. Instead of attaching gates directly to filters, we attach gates to the feature maps after batch normalization (as shown by the red dotted lines). The gate can be generated by propagating the generator networks in either forward or backward direction. Both the original network and the gate generator are trained together as the whole pipeline is fully differentiable. $\{W_1, W_2, W_3, \cdots\}$ are the parameters of the MLP gate generator.

which makes the sample z_{lk} differentiable w.r.t. $\log \alpha_{lk}$. We set $c = 10$ as default, which works well in all our experiments.

Fig. 3 illustrates the overall architecture of Dep-L_0. The original network is shown on the top, and the gate generator (MLP) is shown at the bottom. Here we have a layer-wise partition of the gates, so the gate generator has the same number of layers as the original network. As we discussed in group sparsity, each gate is attached to a filter's output feature map after batch normalization. The input of the gate generator is initialized with a vector of 1's (i.e., $\log \alpha_0 = \mathbf{1}$, such that all the input neurons are activated at the beginning). The values of gates $\boldsymbol{z}$ are generated as we forward propagate the generator. The generated $\boldsymbol{z}$s are then attached to original networks, and the gate dependencies can be learned from the data directly. The whole pipeline (the original network and the gate generator) is fully differentiable as the Hard Concrete distribution (instead of Bernoulli) is used to sample $\boldsymbol{z}$, so that we can use backpropagation to optimize the whole pipeline.

Furthermore, as shown in Fig. 3, the dependencies can be modeled in a backward direction as well, i.e., we generate the gates from the last layer L of MLP first, and then generate the gates from layer $L-1$, and so on. In our experiments, we will evaluate the performance impacts of both forward and backward modeling.

In addition to MLPs, other network architectures such as LSTMs and CNNs can be used to model the gate generator as well. However, neither LSTMs nor CNNs achieves a competitive performance to MLPs in our experiments. Detailed ablation study is provided in the supplementary material[2].

[2] https://arxiv.org/abs/2107.00070

4 Experiments

In this section we compare Dep-L_0 with the state-of-the-art pruning algorithms for CNN architecture pruning. In order to demonstrate the generality of Dep-L_0, we consider multiple image classification benchmarks (CIFAR10, CIFAR100 [23] and ImageNet [5]) and multiple modern CNN architectures (VGG16 [37], ResNet50, and ResNet56 [14]). As majority of the computations of modern CNNs are in the convolutional layers, following the competing pruning methods, we only prune the convolutional filters and leave the fully connected layers intact (even though our method can be used to prune any layers of a network). For a fair comparison, our experiments closely follow the benchmark settings provided in the literature. All our experiments are performed with PyTorch on Nvidia V100 GPUs. The details of the experiment settings can be found in the supplementary material.

L_0-HC implementations From our experiments, we found that the original L_0-HC implementation[3] has a couple issues. First, the binary gates are not properly attached after batch normalization, which results in pruned neurons still having impact after being removed. Second, it only uses one optimizer – Adam for the original network parameters and the hard concrete parameters. We noted that using two optimizers: SGD with momentum for the original network and Adam for the hard concrete parameters works better. Therefore, we fixed these issues of L_0-HC for all the experiments and observed improved performance. For a fair comparison, we follow the same experiment settings as in Dep-L_0, and tune L_0-HC for the best performance.

4.1 CIFAR10 Results

We compare Dep-L_0 with ten state-of-the-art filter pruning algorithms, including our main baseline L_0-HC, in this experiment. Since the baseline accuracies in all the reference papers are different, we compare the performances of all competing methods by their accuracy gains Δ_{Acc} and their pruning rates in terms of FLOPs and network parameters. For Dep-L_0, we evaluate our algorithm with *forward* and *backward* dependency modeling. Table 1 provides the results on CIFAR10. As can be seen, for VGG16, our algorithm (with *backward* dependency modeling) achieves the highest FLOPs reduction of 65.9% on CIFAR10 with only 0.1% of accuracy loss. For ResNet56, our *forward* dependency modeling achieves the highest accuracy gain of 0.2% with a very competitive FLOPs reduction of 45.5%.

Since L_0-HC is our main baseline, we highlight the comparison between Dep-L_0 and L_0-HC in the table. As we can see, Dep-L_0 outperforms L_0-HC consistently in all the experiments. For VGG16, L_0-HC prunes only 39.8% of FLOPs but suffers from a 0.4% of accuracy drop, while our algorithm prunes more (65.9%) and almost keeps the same accuracy (-0.1%). For ResNet56, our algorithm prunes more (45.5% v.s. 44.1%) while achieves a higher accuracy (0.2% vs. -0.5%) than that of L_0-HC.

[3] https://github.com/AMLab-Amsterdam/L0_regularization

Table 1. Comparison of pruning methods on CIFAR10. "Δ_{Acc}": '+' denotes accuracy gain; '-' denotes accuracy loss; the worst result is in red. "FLOPs (P.R. %)": pruning ratio in FLOPs. "Params. (P.R. %)": prune ratio in parameters. "-": results not reported in original paper.

Model	Method	Acc. (%)	Δ_{Acc}	FLOPs (P.R. %)	Params. (P.R. %)
VGG16	Slimming [31]	93.7→93.8	+0.1	195M (51.0)	2.30M (88.5)
	DCP [43]	94.0→94.6	**+0.6**	109.8M (65.0)	**0.94M (93.6)**
	AOFP [8]	93.4→93.8	+0.4	215M (31.3)	-
	HRank [27]	94.0→93.4	-0.6	145M (53.5)	2.51M (82.9)
	L_0-HC (Our implementation)	93.5→93.1	-0.4	135.6M (39.8)	2.8M (80.9)
	Dep-L_0 (forward)	93.5→93.5	0	111.9M (64.4)	2.1M (85.7)
	Dep-L_0 (backward)	93.5→93.4	-0.1	**107.0M (65.9)**	1.8M (87.8)
ResNet56	SFP [15]	93.6→93.4	-0.2	59.4M (53.1)	-
	AMC [17]	92.8→91.9	-0.9	62.5M (50.0)	-
	DCP [43]	93.8→93.8	0	67.1M (47.1)	**0.25M (70.3)**
	FPGM [16]	93.6→93.5	-0.1	59.4M (52.6)	-
	TAS [10]	94.5→93.7	-0.8	**59.5M (52.7)**	-
	HRank [27]	93.3→93.5	**+0.2**	88.7M (29.3)	0.71M (16.8)
	L_0-HC (Our implementation)	93.3→92.8	-0.5	71.0M (44.1)	0.46M (45.9)
	Dep-L_0 (forward)	93.3→93.5	**+0.2**	69.1M (45.5)	0.48M (43.5)
	Dep-L_0 (backward)	93.3→93.0	-0.3	66.7M (47.4)	0.49M (42.4)

Table 2. Comparison of pruning methods on CIFAR100. "Δ_{Acc}": '+' denotes accuracy gain; '-' denotes accuracy loss; the worst result is in red. "FLOPs (P.R. %)": pruning ratio in FLOPs. "Params. (P.R. %)": prune ratio in parameters. "-": results not reported in original paper.

Model	Method	Acc. (P.R. %)	Δ_{Acc}	FLOPs (P.R. %)	Params. (%)
VGG16	Slimming [31]	73.3→73.5	+0.2	250M (37.1)	5.0M (75.1)
	L_0-HC (Our implementation)	72.2→70.0	-1.2	138M (56.2)	4.1M (72.5)
	Dep-L_0 (forward)	72.2→71.6	-0.6	**98M (68.8)**	**2.1M (85.7)**
	Dep-L_0 (backward)	72.2→72.5	**+0.3**	105M (66.6)	2.2M (85.0)
ResNet56	SFP [15]	71.4→68.8	-2.6	**59.4M (52.6)**	-
	FPGM [16]	71.4→69.7	-1.7	**59.4M (52.6)**	-
	TAS [10]	73.2→72.3	-0.9	61.2M (51.3)	-
	L_0-HC (Our implementation)	71.8→70.4	-1.4	82.2M (35.2)	0.73M (15.2)
	Dep-L_0 (forward)	71.8→71.7	**-0.1**	87.6M (30.9)	0.56M (34.9)
	Dep-L_0 (backward)	71.8→71.2	-0.6	93.4M (26.3)	**0.52M (39.5)**

4.2 CIFAR100 Results

Experimental results on CIFAR100 are reported in Table 2, where Dep-L_0 is compared with four state-of-the-arts pruning algorithms: Slimming [31], SFP [15], FPGM [16] and TAS [10]. Similar to the results on CIFAR10, on this benchmark Dep-L_0 achieves the best accuracy gains and very competitive or sometimes even higher prune rates compared to the state-of-the-arts. More importantly, Dep-L_0 outperforms L_0-HC in terms of classification accuracies and pruning rates consistently, demonstrating the effectiveness of dependency modeling.

Table 3. Comparison of pruning methods on ImageNet. "Δ_{Acc}": '+' denotes accuracy gain; '-' denotes accuracy loss; the worst result is in red. "FLOPs (P.R. %)": pruning ratio in FLOPs. "Params. (P.R. %)": prune ratio in parameters. "-": results not reported in original paper.

Model	Method	Acc. (%)	Δ_{Acc}	FLOPs (P.R.%)	Params. (P.R.%)
ResNet50	SSS-32 [20]	76.12 → 74.18	-1.94	2.82B (31.1)	18.60M (27.3)
	DCP [43]	76.01 → 74.95	-1.06	**1.82B (55.6)**	**12.40M (51.5)**
	GAL-0.5 [29]	76.15 → 71.95	-4.2	2.33B (43.1)	21.20M (17.2)
	Taylor-72 [35]	76.18 → 74.50	-1.68	2.25B (45.0)	14.20M (44.5)
	Taylor-81 [35]	76.18 → 75.48	-0.70	2.66B (34.9)	17.90M (30.1)
	FPGM-30 [16]	76.15 → 75.59	-0.56	2.36B (42.2)	-
	FPGM-40 [16]	76.15 → 74.83	-1.32	1.90B (53.5)	-
	LeGR [3]	76.10 → 75.70	-0.40	2.37B (42.0)	-
	TAS [10]	77.46 → 76.20	-1.26	2.31B (43.5)	-
	HRank [27]	76.15 → 74.98	-1.17	2.30B (43.8)	16.15M (36.9)
	L_0-HC (Our implementation)	76.15 → 76.15	0	4.09B (0.00)	25.58M (0.00)
	Dep-L_0 (forward)	76.15 → 74.77	-1.38	2.58B (36.9)	16.04M (37.2)
	Dep-L_0 (backward)	76.15 → 74.70	-1.45	2.53B (38.1)	14.34M (43.9)

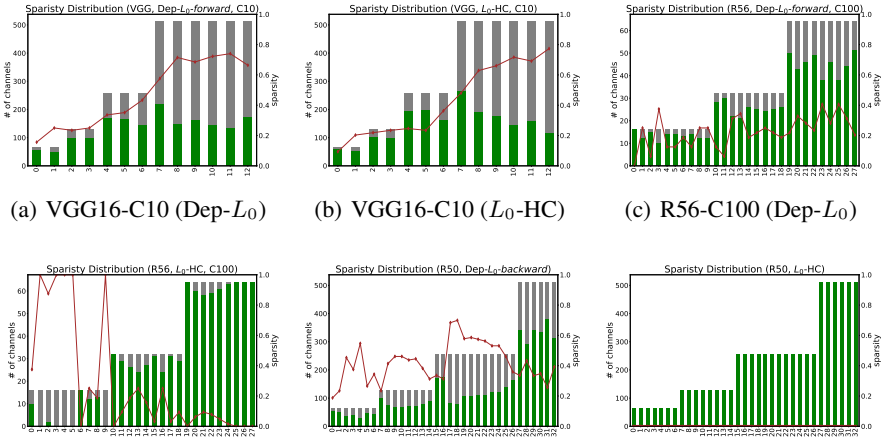
4.3 ImageNet Results

The main goal of the paper is to make L_0-HC once again competitive on the large-scale benchmark of ImageNet. In this section, we conduct a comprehensive experiment on ImageNet, where the original L_0-HC fails to prune without a significant damage of model quality [11]. Table 3 reports the results on ImageNet, where eight state-of-the-art filter pruning methods are included, such as SSS-32 [20], DCP [43], Taylor [35], FPGM [16], HRank [27] and others. As can be seen, Dep-L_0 (forward) prunes 36.9% of FLOPs and 37.2% of parameters with a 1.38% of accuracy loss, which is comparable with other state-of-the-art algorithms as shown in the table.

Again, since L_0-HC is our main baseline, we highlight the comparison between Dep-L_0 and L_0-HC in the table. We tune the performance of L_0-HC extensively by searching for the best hyperparameters in a large space. However, even with extensive efforts, L_0-HC still fails to prune the network without a significant damage of model quality, confirming the observation made by [11]. On the other hand, our Dep-L_0 successfully prunes ResNet50 with a very competitive pruning rate and high accuracy compared to the state-of-the-arts, indicating that our dependency modeling indeed makes the original L_0-HC very competitive on the large-scale benchmark of ImageNet – the main goal of the paper.

4.4 Study of Learned Sparse Structures

To understand of the behavior of Dep-L_0, we further investigate the sparse structures learned by Dep-L_0 and L_0-HC, with the results reported in Fig. 4. For VGG16 on CIFAR10, Figs. 4(a-b) demonstrate that both algorithms learn a similar sparsity pattern: the deeper a layer is, the higher prune ratio is, indicating that the shallow layers of VGG16 are more important for its predictive performance. However, for deeper networks such as ResNet56 and ResNet50, the two algorithms perform very differently.



(a) VGG16-C10 (Dep-L_0) (b) VGG16-C10 (L_0-HC) (c) R56-C100 (Dep-L_0)

(d) R56-C100 (L_0-HC) (e) R50-ImageNet (Dep-L_0) (f) R50-ImageNet (L_0-HC)

Fig. 4. The layer-wise prune ratios (red curves) of learned sparse structures. The height of a bar denotes the number of filters of a convolutional layer and gray (green) bars correspond to the original (pruned) architecture, respectively. "R50/R56": ResNet 50/56; "C10/C100": CIFAR10/100.

For ResNet56 on CIFAR100, Figs. 4(c-d) show that Dep-L_0 sparsifies each layer by a roughly similar prune rate: it prunes around 20% of the filters in first 12 layers, and around 30% of the filters in the rest of layers. However, on the same benchmark L_0-HC tends to prune *all or nothing*: it completely prunes 5 out of 28 layers, but does not prune any filters in other six layers; for the rest of layers, the sparsity produced by L_0-HC is either extremely high or low. As of ResNet50 on ImageNet, Figs. 4(e-f) show that the difference between Dep-L_0 and L_0-HC is more significant: Dep-L_0 successfully prunes the model with a roughly similar prune rate across all convolutional layers, while L_0-HC fails to prune any filters.

4.5 Run-time Comparison

The main architectural difference between Dep-L_0 and L_0-HC is the gate generator. Even though the gate generator (MLP) is relatively small compared to the original deep network to be pruned, its existence increases the computational complexity of Dep-L_0. Thus, it is worth comparing the run-times of Dep-L_0 and L_0-HC as well as their convergence rates in terms of sparse structure search. Once a sparse structure is learned by a pruning algorithm, we can extract the sparse network from the original network and continue the training on the smaller structure such that we can reduce the total time to solution (TTS). To this end, we compare Dep-L_0 with L_0-HC in terms of (1) structure search convergence rate, i.e., how many training epochs are needed for a pruning algorithm to converge to a sparse structure? (2) Per-epoch training time before convergence, and (3) the total time to solution (TTS). The results are reported in Table 4. As can be seen, Dep-L_0 (both *forward* and *backward*) converges to a sparse structure in roughly half of the epochs that L_0-HC needs (column 4). Even though the per-epoch training

Table 4. Run-time comparison between Dep-L_0 and L_0-HC. "R50/R56": ResNet50/56; "C10/C100": CIFAR10/100. "BC": Before Convergence; "TTS": Time to Solution.

Benchmark	Method	# Epochs	# Epochs BC	Per-epoch Time BC	TTS
R56-C10	L_0-HC	300	218	37.9 sec	160 min
	Dep-L_0 (forward)	300	**106**	42.2 sec	**124 min**
	Dep-L_0 (backward)	300	140	42.6 sec	141 min
R56-C100	L_0-HC	300	117	38.7 sec	167 min
	Dep-L_0 (forward)	300	**58**	43.9 sec	**127 min**
	Dep-L_0 (backward)	300	61	43.6 sec	133 min
R50-ImageNet	L_0-HC	90	fail to prune	4185 sec	104.6 h
	Dep-L_0 (forward)	90	**30**	4342 sec	**59.5 h**
	Dep-L_0 (backward)	90	32	4350 sec	60.1 h

time of Dep-L_0 is 12% (4%) larger than that of L_0-HC on CIFAR10/100 (ImageNet) due to the extra computation of the gate generator (column 5), the total time to solution reduces by 22.5% (43.1%) on the CIFAR10/100 (ImageNet) benchmarks thanks to the faster convergence rates and sparser models induced by Dep-L_0 as compared to L_0-HC (column 6).

5 Conclusion and Future Work

We propose Dep-L_0, an improved L_0 regularized network sparsification algorithm via dependency modeling. The algorithm is inspired by a recent observation of Gale et al. [11] that L_0-HC performs inconsistently in large-scale learning tasks. Through the lens of variational inference, we found that this is likely due to the mean-field assumption in variational inference that ignores the dependency among all the neurons for network pruning. We further propose a dependency modeling of binary gates to alleviate the deficiency of the original L_0-HC. A series of experiments are performed to evaluate the generality of our Dep-L_0. The results show that our Dep-L_0 outperforms the original L_0-HC in all the experiments consistently, and the dependency modeling makes the L_0-based sparsification once again very competitive and sometimes even outperforms the state-of-the-art pruning algorithms. Further analysis shows that Dep-L_0 also learns a better structure in fewer epochs, and reduces the total time to solution by 20%-40%.

As for future work, we plan to explore whether dependency modeling can be used to improve other pruning methods. To the best of our knowledge, there are very few prior works considering dependency for network pruning (e.g., [36]). Our results show that this may be a promising direction to further improve many existing pruning algorithms. Moreover, the way we implement dependency modeling is still very preliminary, which can be improved further in the future.

Acknowledgment

We would like to thank the anonymous reviewers for their comments and suggestions, which helped improve the quality of this paper. We would also gratefully acknowledge the support of VMware Inc. for its university research fund to this research.

References

1. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer (2007)
2. Blei, D.M., Kucukelbir, A., McAuliffe, J.D.: Variational inference: A review for statisticians. Journal of the American Statistical Association pp. 859–877 (2017)
3. Chin, T.W., Ding, R., Zhang, C., Marculescu, D.: Legr: Filter pruning via learned global ranking. arXiv pp. arXiv–1904 (2019)
4. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. arXiv preprint arXiv:1602.02830 (2016)
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR. pp. 248–255 (2009)
6. Deng, L., Li, G., Han, S., Shi, L., Xie, Y.: Model compression and hardware acceleration for neural networks: A comprehensive survey. In: Proceedings of the IEEE. vol. 108, pp. 485–532 (2020)
7. Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in neural information processing systems. pp. 1269–1277 (2014)
8. Ding, X., Ding, G., Guo, Y., Han, J., Yan, C.: Approximated oracle filter pruning for destructive cnn width optimization. arXiv preprint arXiv:1905.04748 (2019)
9. Ding, X., Ding, G., Zhou, X., Guo, Y., Han, J., Liu, J.: Global sparse momentum sgd for pruning very deep neural networks. In: Advances in Neural Information Processing Systems (2019)
10. Dong, X., Yang, Y.: Network pruning via transformable architecture search. In: Advances in Neural Information Processing Systems. pp. 760–771 (2019)
11. Gale, T., Elsen, E., Hooker, S.: The state of sparsity in deep neural networks. arXiv preprint arXiv:1902.09574 (2019)
12. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: International Conference on Machine Learning. pp. 1737–1746 (2015)
13. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems. pp. 1135–1143 (2015)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. pp. 770–778 (2016)
15. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. arXiv preprint arXiv:1808.06866 (2018)
16. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4340–4349 (2019)
17. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: Amc: Automl for model compression and acceleration on mobile devices. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 784–800 (2018)
18. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
19. Hu, H., Peng, R., Tai, Y.W., Tang, C.K.: Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250 (2016)
20. Huang, Z., Wang, N.: Data-driven sparse structure selection for deep neural networks. In: Proceedings of the European conference on computer vision (ECCV). pp. 304–320 (2018)
21. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. International Conference on Machine Learning (ICML) (2015)

22. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866 (2014)
23. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
24. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in neural information processing systems. pp. 598–605 (1990)
25. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710 (2016)
26. Li, Y., Ji, S.: l_0-arm: Network sparsification via stochastic binary optimization. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 432–448. Springer (2019)
27. Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., Shao, L.: Hrank: Filter pruning using high-rank feature map. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1529–1538 (2020)
28. Lin, S., Ji, R., Li, Y., Deng, C., Li, X.: Toward compact convnets via structure-sparsity regularized filter pruning. IEEE transactions on neural networks and learning systems **31**(2), 574–588 (2019)
29. Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F., Doermann, D.: Towards optimal structured cnn pruning via generative adversarial learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2790–2799 (2019)
30. Liu, B., Wang, M., Foroosh, H., Tappen, M., Pensky, M.: Sparse convolutional neural networks. In: CVPR. pp. 806–814 (2015)
31. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2736–2744 (2017)
32. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization. International Conference on Learning Representations (ICLR) (2018)
33. Mitchell, T.J., Beauchamp, J.J.: Bayesian variable selection in linear regression. Journal of the american statistical association **83**(404), 1023–1032 (1988)
34. Molchanov, D., Ashukha, A., Vetrov, D.: Variational dropout sparsifies deep neural networks. arXiv preprint arXiv:1701.05369 (2017)
35. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 11264–11272 (2019)
36. Park, S., Lee, J., Mo, S., Shin, J.: Lookahead: a far-sighted alternative of magnitude-based pruning. arXiv preprint arXiv:2002.04809 (2020)
37. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
38. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in neural information processing systems (2016)
39. Ye, J., Lu, X., Lin, Z., Wang, J.Z.: Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. arXiv preprint arXiv:1802.00124 (2018)
40. You, Z., Yan, K., Ye, J., Ma, M., Wang, P.: Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 2133–2144 (2019)
41. Zagoruyko, S., Komodakis, N.: Wide residual networks (2016)
42. Zhu, M., Gupta, S.: To prune, or not to prune: exploring the efficacy of pruning for model compression. arXiv preprint arXiv:1710.01878 (2017)
43. Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., Zhu, J.: Discrimination-aware channel pruning for deep neural networks. In: Advances in Neural Information Processing Systems. pp. 875–886 (2018)