Neuron-level Structured Pruning using Polarization Regularizer

Tao Zhuang[1], **Zhixuan Zhang**[*1], **Yuheng Huang**[2], **Xiaoyi Zeng**[1], **Kai Shuang**[2], **Xiang Li**[1]
[1]Alibaba Group
[2]Beijing University of Posts and Telecommunications
{zhuangtao.zt, zhibing.zzx}@alibaba-inc.com, hyhlryf@bupt.edu.cn,
yuanhan@taobao.com, shuangk@bupt.edu.cn, leo.lx@alibaba-inc.com

Abstract

Neuron-level structured pruning is a very effective technique to reduce the computation of neural networks without compromising prediction accuracy. In previous works, structured pruning is usually achieved by imposing L1 regularization on the scaling factors of neurons, and pruning the neurons whose scaling factors are below a certain threshold. The reasoning is that neurons with smaller scaling factors have weaker influence on network output. A scaling factor close to 0 actually suppresses a neuron. However, L1 regularization lacks discrimination between neurons because it pushes all scaling factors towards 0. A more reasonable pruning method is to only suppress unimportant neurons (with 0 scaling factors), and simultaneously keep important neurons intact (with larger scaling factor). To achieve this goal, we propose a new regularizer on scaling factors, namely *polarization* regularizer. Theoretically, we prove that polarization regularizer pushes some scaling factors to 0 and others to a value $a > 0$. Experimentally, we show that structured pruning using polarization regularizer achieves much better results than using L1 regularizer. Experiments on CIFAR and ImageNet datasets show that polarization pruning achieves the state-of-the-art result.

1 Introduction

Network pruning is proved to effectively reduce the computational cost of inference without significantly compromising accuracy [Liu et al., 2019a]. There are two major branches of network pruning. One branch is *unstructured* pruning, which prunes at the level of individual weights [LeCun et al., 1989, Han et al., 2015, Zhou et al., 2019, Ding et al., 2019, Frankle and Carbin, 2019]. The other branch is *structured* pruning, which prunes at the level of neurons (or channels) [Wen et al., 2016, Liu et al., 2017, Ye et al., 2018]. Although unstructured pruning usually reduces more weights than structured pruning [Liu et al., 2019a], it has the drawback that the resulting weight matrices are sparse, which cannot lead to speedup without dedicated hardware or libraries [Han et al., 2016]. In this paper, we focus on neuron-level structured pruning, which does not require additional hardware or libraries to reduce computation on common GPU/CPU devices.

For structured pruning, one promising method is to associate each neuron with a scaling factor, and regularize these scaling factors in training. Then neurons with scaling factors below a certain threshold are pruned [Huang and Wang, 2018]. The regularizer on scaling factors is usually chosen to be L1 [Liu et al., 2017]. However, L1 regularizer tries to push all scaling factors to 0. It is often difficult to find a reasonable pruning threshold. For example, the distribution of scaling factors under L1 regularization for VGG-16 network trained on CIFAR-10 dataset is shown in Figure 1 (a), where

[*]Corresponding author

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

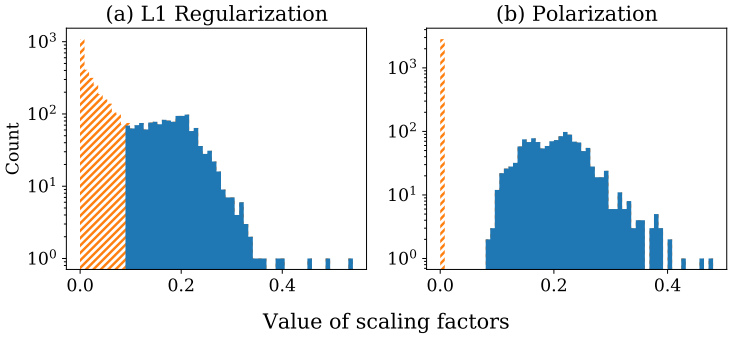


Figure 1: The distributions of scaling factors in VGG-16 trained on CIFAR-10 dataset, with L1 and polarization regularizers respectively. Under the same pruning ratio for both regularizers, the orange part are pruned.

the scaling factors distribute densely around the threshold value. A cut at the threshold value is not very reasonable because there is no margin around the threshold to separate the pruned neurons from the preserved ones. Pruning with this threshold will lead to severe accuracy drop. A more reasonable regularizer should separate the pruned and the preserved neurons more obviously, with a larger margin between them. To attain this goal, we propose a novel regularizer named *polarization*. Different from L1 regularizer that pushes all scaling factors to 0, polarization simultaneously pushes a proportion of scaling factors to 0 (thus pruning these neurons), and the rest scaling factors to a value larger than 0 (thus preserving these neurons). Intuitively, instead of suppressing all neurons in pruning, polarization tries to suppress only a proportion of neurons while keeps others intact. Polarization regularizer naturally makes a distinction between pruned and preserved neurons. And the resulted scaling factors are more separable. As shown in Figure 1 (b), polarization leads to an obvious margin between scaling factors of the pruned neurons (the orange part) and the preserved ones (the blue part). Pruning using polarization is more reasonable because the pruned neurons has much smaller influence on network output than the preserved neurons. Our code is available at `https://github.com/polarizationpruning/PolarizationPruning`.

We summarize our contributions as follows:

- We propose a novel regularizer, namely polarization, for structured pruning of neural networks. We theoretically analyzed the properties of polarization regularizer and proved that it simultaneously pushes a proportion of scaling factors to 0 and others to values larger than 0.
- We verify the effectiveness of polarization pruning on the widely-used CIFAR and ImageNet datasets, and achieve state-of-the-art pruning results.

2 Related Work

Neuron-level structured pruning is also called channel/filter pruning in works focusing on convolutional neural network (CNN) structures. It is equivalent to unstructured pruning with the constraint that all weights connected to one neuron must be pruned or preserved together. There are usually three stages for structured pruning [Liu et al., 2019a]: 1) train an over-parameterized model, 2) prune the trained model according to some criteria, and 3) fine-tune the pruned model to boost performance. Structured pruning methods are characterized by how to handle these 3 stages, especially the training stage. Some methods [Li et al., 2017, He et al., 2018a, 2019] do not add extra sparsity regularizer on the weights of networks in the training stage, while other works [Wen et al., 2016, Zhou et al., 2016, Alvarez and Salzmann, 2016, Lebedev and Lempitsky, 2016, He et al., 2017] impose group sparsity regularization on network weights during training. Yang et al. [2020] propose to use the Hoyer regularizer for pruning. Peng et al. [2019] exploit the inter-channel dependency to determine the optimal combination of preserved channels in the pre-trained model. Luo and Wu [2020] insert binary mask layers after convolution layers during training and removes filters corresponding to zero mask values. Zhuang et al. [2018] fine-tune the pre-trained model with discrimination-aware losses to choose the channels of each layer in the training stage. A group of sparsity regularization pruning

methods introduce a scaling factor for each neuron and add sparsity regularization on the scaling factors in training. Then neurons are pruned based on the values of their scaling factors. Liu et al. [2017] and Ye et al. [2018] use the scale factor in Batch Normalization (BN) as the scaling factor for each neuron, whereas Huang and Wang [2018] add a new scaling factor for each neuron and thus does not depend on BN anymore. During training, L1 regularizer is imposed on the scaling factors of neurons to induce sparsity [Liu et al., 2017, Ye et al., 2018]. After training, neurons with the smallest scaling factors are pruned. Our method falls into this group. And the work most related to ours is [Liu et al., 2017]. We also use the scale factor in BN as the scaling factor of a neuron. But different from all previous works, we propose a novel polarization regularizer to make the pruned and preserved neurons more separable.

Another remotely related area is Neural Architecture Search (NAS), which is employed to find computationally efficient architectures [He et al., 2018b, Liu et al., 2019b]. And some works [Cai et al., 2020] requires much more computation in the training and search processes. Our method is a sparsity regularization pruning method. So we do not focus on comparisons with these NAS methods in this paper.

3 Structured Pruning through Polarization

3.1 Preliminaries

Given a training set $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, where $\boldsymbol{x}_i, \boldsymbol{y}_i$ denote the input feature and the label of sample i respectively, we need to train a neural network $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ denotes the parameters of the network. We introduce a scaling factor for each neuron and represent the scaling factors as a vector $\boldsymbol{\gamma} \in \mathbf{R}^n$, where n is the number of neurons in the network. As in [Liu et al., 2017], we use the scale factor in BN as the scaling factor of each neuron, because of the wide adoption of BN in modern neural networks. The objective function for network training with regularization on scaling factors is:

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{f}(\boldsymbol{x_i}; \boldsymbol{\theta}), \boldsymbol{y}_i) + R(\boldsymbol{\theta}) + \lambda R_s(\boldsymbol{\gamma}) \tag{1}$$

where $L(\cdot)$ is the loss function, $R(\cdot)$ is usually the L2 regularization on weights of the network, and $R_s(\cdot)$ is the sparsity regularizer on scaling factors of neurons. In pruning, a threshold is chosen and neurons with scaling factors below the threshold are pruned. In [Liu et al., 2017], the sparsity regularizer is chosen to be L1, i.e. $R_s(\boldsymbol{\gamma}) = \|\boldsymbol{\gamma}\|_1$. The effect of L1 regularization is to push all scaling parameters to 0. Therefore, L1 regularization lacks discrimination between pruned and preserved neurons. A more reasonable pruning method is to only suppress unimportant neurons (with 0 scaling factor) and simultaneously keep important neurons intact (with larger scaling factor). To achieve this goal, we propose a new regularizer on scaling factors, namely *polarization* regularizer.

3.2 Polarization Regularizer

Let $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_n)$, and $\mathbf{1}_n = (1, 1, \cdots, 1) \in \mathbf{R}^n$. Let $\bar{\gamma}$ denote the mean of $\gamma_1, \ldots, \gamma_n$:

$$\bar{\gamma} := \frac{1}{n} \mathbf{1}_n^\top \boldsymbol{\gamma} = \frac{1}{n} \sum_{i=1}^{n} \gamma_i$$

γ_i is the scaling factor for neuron i. Obviously, a scaling factor should be positive and bounded. So it is reasonable to constrain each scaling factor to a range: $\gamma_i \in [0, a]$, where $a > 0$. L1 regularizer pushes all scaling factors to 0, because the optimal solution of $\min_{\boldsymbol{\gamma} \in [0,a]^n} \|\boldsymbol{\gamma}\|_1$ is $\mathbf{0}$. To obtain the polarization effect, we need to prevent the scaling factors from converging to one value. So we define the polarization regularizer to be:

$$\begin{aligned} R_s(\boldsymbol{\gamma}) &= t\|\boldsymbol{\gamma}\|_1 - \|\boldsymbol{\gamma} - \bar{\gamma}\mathbf{1}_n\|_1 \\ &= \sum_{i=1}^{n} t|\gamma_i| - |\gamma_i - \bar{\gamma}|, (t \in \mathbb{R}, \gamma_i \in [0, a]) \end{aligned} \tag{2}$$

In Equation (2), we add the new term $-\|\boldsymbol{\gamma} - \bar{\gamma}\mathbf{1}_n\|_1$ to the L1 term $\|\boldsymbol{\gamma}\|_1$. The effect of $-\|\boldsymbol{\gamma} - \bar{\gamma}\mathbf{1}_n\|_1$ is to separate $\gamma_i, 1 \le i \le n$ from their average as far as possible. Actually, $-\|\boldsymbol{\gamma} - \bar{\gamma}\mathbf{1}_n\|_1$ reaches its

maximum value when all γ_i are equal, and reaches its minimum value when half of $\gamma_i, 1 \leq i \leq n$ equals 0, while the other half equals a. We also use a hyper-parameter t to control the weight of $\|\boldsymbol{\gamma}\|_1$ relative to $-\|\boldsymbol{\gamma} - \bar{\gamma}\mathbf{1}_n\|_1$. As we will show later, t also controls the proportion of scaling factors that equal 0 under polarization regularizer.

The polarization regularizer has several nice properties. The first property is *permutation invariance*: Let $\pi(\boldsymbol{\gamma})$ denote the vector obtained by an arbitrary permutation of $\boldsymbol{\gamma}$ on its n dimensions. Then it is obvious that $R_s(\pi(\boldsymbol{\gamma})) = R_s(\boldsymbol{\gamma})$, which means that the polarization regularizer is permutation invariant. This property ensures all neurons be equally treated in pruning: no prior pruning bias for any neuron. The second property is concavity, as stated in Lemma 3.1:

Lemma 3.1. *$R_s(\boldsymbol{\gamma})$ defined in Equation (2) is concave.*

To see the effect of polarization regularizer more clearly, we need to solve the following minimization problem:

$$\min_{\boldsymbol{\gamma} \in [0,a]^n} R_s(\boldsymbol{\gamma}) = \sum_{i=1}^{n} t|\gamma_i| - |\gamma_i - \bar{\gamma}|, (t \in \mathbb{R}) \tag{3}$$

This is a minimization of $R_s(\boldsymbol{\gamma})$ on an n-dimensional cube. Utilizing the concavity of $R_s(\boldsymbol{\gamma})$, we prove that the optimal value of Equation (3) is attained on vertices of the n-dimensional cube $[0, a]^n$. It is well known that an n-dimensional cube has 2^n vertices. Utilizing the permutation invariance property of $R_s(\boldsymbol{\gamma})$, we finally prove the following theorem:

Theorem 3.2. *A class of optimal solutions of Equation (3) is that either $\lfloor n\rho \rfloor$ or $\lfloor n\rho \rfloor + 1$ number of $\gamma_i, (1 \leq i \leq n)$ are a, and the rest are 0, where:*

$$\rho = \begin{cases} -t/4 + 1/2, & -2 \leq t \leq 2 \\ 0, & t > 2 \\ 1, & t < -2 \end{cases} \tag{4}$$

The detailed proof of Lemma 3.1 and Theorem 3.2 is in Appendix A. Theorem 3.2 states that the effect of polarization regularizer is to push a proportion of scaling factors to 0 and the rest scaling factors to a. The proportion ρ depends piecewise linearly on the hyper-parameter t as in Equation (4). Therefore, Theorem 3.2 precisely shows the effect of polarization regularizer, and tells that the proportion of 0s is determined by t.

As for the choice of a scaling factor for a neuron, it is convenient to reuse the scale factor in BN of this neuron if it has BN. It is also feasible to introduce an extra scaling factor on the output of a neuron as in [Huang and Wang, 2018]. Our polarization regularizer is applicable for both choices of scaling factors above. To fully separate the two poles, the upper bound a of scaling factors should not be too small, especially when reusing the scale factors in BN. The reason is as follows. Let $z_{in}^{(i)}$ and $z_{out}^{(i)}$ be the input and output of BN on neuron i, then BN performs the following transformation:

$$\hat{z} = \frac{z_{in}^{(i)} - \mathrm{E}[z_{in}^{(i)}]}{\sqrt{\mathrm{Var}[z_{in}^{(i)}]}}$$
$$z_{out}^{(i)} = \gamma_i \hat{z} + \beta_i$$

where $\mathrm{E}[z_{in}^{(i)}]$ and $\mathrm{Var}[z_{in}^{(i)}]$ are the mean and variance of $z_{in}^{(i)}$ respectively. As pointed out in [Ioffe and Szegedy, 2015], it is important for BN to include the *identity transform* by setting $\gamma_i = \sqrt{\mathrm{Var}[z_{in}^{(i)}]}$ and $\beta_i = \mathrm{E}[z_{in}^{(i)}]$. Therefore, a should be sufficiently large such that $\forall 1 \leq i \leq n$, $a > \sqrt{\mathrm{Var}[z_{in}^{(i)}]}$.

Implementation Details In the training stage, we optimize Equation (1) using Stochastic Gradient Descent (SGD). Due to the L1 norm in $R_s(\cdot)$, there exist nondifferentiable points in Equation (1). We use subgradients on these nondifferentiable points. To implement the constraint that $\gamma_i \in [0, a]$, we simply clamp the value of γ_i to a when it is larger than a, and clamp it to 0 when it is smaller than 0 during the training process. All scaling factors are initialized to be 0.5 in image classification tasks as in [Liu et al., 2017]. We find that in our neural networks, the standard deviation of input to BN is mostly smaller than 1. Therefore, we set the upper bound of scaling factor a to 1 in this paper. Empirically, we find that the pruning result is insensitive to the exact value of a.

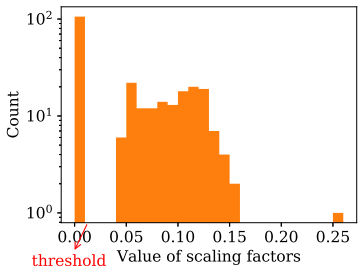


Figure 2: Our strategy to determine the pruning threshold.

3.3 Pruning Strategy

After training with the polarization regularizer, we obtain the values and distribution of scaling factors. We still need a threshold value to prune away neurons with small scaling factors. In [Liu et al., 2017], the threshold is determined by the percentile of neurons to be pruned. Although this method precisely controls the percentile of neurons to be pruned, it often leads to an unreasonable threshold, as shown in Figure 1 (a). We utilize the polarization effect and propose a more reasonable strategy to automatically set the threshold. Because of the polarization effect, the distribution graph always has at least two local maximums (peaks): one is centered near 0, and others are centered around larger values, as is shown in Figure 2. Our strategy is to only prune the neurons that belong to the peak nearest to 0. So the threshold is located at the tail of the peak nearest to 0, as shown by the example in Figure 2. More specifically, when we draw the distribution histogram, we set the bin width to 0.01. Then we scan the bins in the histogram from left to right and find the first local minimum bin. Then the horizontal coordinate of this bin is the threshold for pruning. Using this strategy, we automatically prune the neurons with scaling factors nearest to 0. And empirically we find that this strategy produces good pruning results very robustly.

In our method, we control the number of reduced Floating-point Operations (FLOPs) by adjusting two hyper-parameters: λ in Equation (1) and t in Equation (2). Given a targeted number of reduced FLOPs, our aim is to choose values of λ and t so that polarization pruning will actually reduce the targeted number of FLOPs. Note that the number of reduced FLOPs is positively related to the number of pruned neurons. Theorem 3.2 gives the equation on how the ratio of 0 scaling factors, which equals $1-\rho$, is determined by hyper-parameter t under polarization regularizer alone. However, when minimizing the training objective in Equation (1), where polarization regularizer is just one part of the objective, there is no theory to quantitatively describe the relation between ρ and t. It is obvious that the hyper-parameter λ in Equation (1) also affects the number of reduced FLOPs. We will present experiment results on the effects of the hyper-parameters λ and t in Section 4.4.

Given a targeted reduced FLOPs, we use a search strategy to determine the hyper-parameters λ and t. We first empirically set λ and t to a range and a value, and then proceed by line search until we are close to the targeted FLOPs. The search algorithm aims to find hyper-parameters that lead to a FLOPs reduction $F \in (F^* - \delta, F^* + \delta)$, where F^* is the target reduction and δ is a permissible range. Our algorithm first performs a coarse search on λ, then carries out a fine search on t, as described in Algorithm 1. In the image classification experiments in this paper, the permissible range δ is set to 5%. Empirically, our algorithm needs to try out about 5 groups of hyper-parameters to stop. For each group of hyper-parameters we only need to train 30 epochs on ImageNet dataset for the FLOPs reduction to be stable, rather than the 120 epochs in full training. So the computational overhead of our search is not too much in practice. Note that Algorithm 1 is empirically designed for the image classification tasks in this paper. Strictly speaking, lines 11-18 in Algorithm 1 have a risk of being an infinite loop. However, empirically Algorithm 1 works well and efficiently in our experiments.

After pruning, we fine-tune the pruned network on the training data.

4 Experiments

In this section, we carried out extensive experiments to evaluate our approach on the image classification task, which is the most widely-used task to test pruning methods. We presents the our main

Algorithm 1 Search for hyper-parameters of Polarization Pruning

Input: Target FLOPs reduction F^* with a permissible range δ.
Output: Hyper-parameter values (λ, t), such that, after pruning, the FLOPs reduction $F \in (F^* - \delta, F^* + \delta)$.

Initialize $(\lambda_l, \lambda_u, t) = (0, 2.0 \times 10^{-4}, 1.2)$
// Fix t and search for λ such that $|F - F^*| < 2\delta$ using binary search.
do
  $\lambda \leftarrow (\lambda_l + \lambda_u)/2$;
  Train network with (λ, t) until the corresponding FLOPs reduction F is stable;
  if $F - F^* \geq 2\delta$ **then**
    $\lambda_u \leftarrow \lambda$;
  else if $F - F^* \leq -2\delta$ **then**
    $\lambda_l \leftarrow \lambda$;
  end
while $|F - F^*| \geq 2\delta$;
// Fix λ and search for t such that $|F - F^*| < \delta$.
while $|F - F^*| \geq \delta$ **do**
  if $F - F^* \geq \delta$ **then**
    $t \leftarrow t - 0.1$
  else
    $t \leftarrow t + 0.1$
  end
  Train network with (λ, t) until the corresponding FLOPs reduction F is stable;
end

experiment results in Section 4.3. In Section 4.4, we discuss the effect of hyper-parameters in our approach. To compare the effects of L1 and polarization regularizers, we visualize the distributions of scaling factors in the training process. We also compare the distributions of the scale factors in BN induced by different regularizers with that of the baseline model. At last, we experiment on a wider spectrum of FLOPs reduction.

4.1 Datasets and Compared Methods

As pointed out in [Gale et al., 2019], the same pruning method may behave differently on small datasets and large datasets. Therefore, we evaluate our method on both small datasets (CIFAR 10/100 [Krizhevsky et al., 2009]) and a large dataset (ImageNet [Russakovsky et al., 2015]). We also experiment with three widely-used deep CNN structures: the VGG-Net [Simonyan and Zisserman, 2015], ResNet [He et al., 2016], and MobileNet V2 [Sandler et al., 2018]. This also makes difficulties for us to compare with previous works, because many works only performed experiment in one dataset, or on one network structure. Therefore, on each dataset and network structure, we only compare with those works that have published experiment results on this dataset and network structure. For a method that released codes, we also run it in our experimental setting. If we get better results for this method than the originally published ones, we will use the better results for this method in our comparisons, and append a "(Our-impl.)" suffix for this method in our result tables. Otherwise we will use the originally published results for this method. We have reviewed all the compared methods in Section 2. Some methods, such as CCP-AC in [Peng et al., 2019], insert additional auxiliary classifiers into the network. As pointed out in [Peng et al., 2019], it is unfair to directly compare these methods with other methods. So we do not compare with these methods in our experiments.

4.2 Experimental Setup

The base VGG model is implemented following [Liu et al., 2017], which add BN to the fully connected layers of VGG. The base ResNet model is implemented following [He et al., 2016]. Specifically, our code implementation is based on PyTorch and Torchvision [Paszke et al., 2019]. We list the detailed parameters in our training in Appendix B. We adjust the hyper-parameters λ and t in our polarization regularizer to control the reduced FLOPs as described in Section 3.3. For ResNet and MobileNet V2, we prune blocks with residual connections as [He et al., 2019], where the last

Table 1: Results on CIFAR-10 and CIFAR-100. Best results are bolded.

Dataset	Model	Approach	Baseline	Pruned		
			Acc. (%)	Acc. (%)	Acc. Drop (%)	FLOPs Reduction
CIFAR-10	ResNet-56	NS [Liu et al., 2017] (Our-impl.)	93.80	93.27	0.53	48%
		CP [He et al., 2017]	92.80	91.80	1.00	50%
		AMC [He et al., 2018b]	92.80	91.90	0.90	50%
		DCP [Zhuang et al., 2018]	93.80	93.49	0.31	50%
		DCP-adapt [Zhuang et al., 2018]	93.80	93.81	-0.01	47%
		SFP [He et al., 2018a]	93.59	93.35	0.24	51%
		FPGM [He et al., 2019]	93.59	93.49	0.10	53%
		CCP [Peng et al., 2019]	93.50	93.46	0.04	47%
		DeepHoyer [Peng et al., 2019] (Our-impl.)	93.80	93.54	0.26	48%
		Ours	93.80	93.83	**-0.03**	47%
	VGG-16	FPGM [He et al., 2019]	93.58	93.54	0.04	34%
		NS [Liu et al., 2017] (Our-impl.)	93.88	93.62	0.26	51%
		Ours	93.88	93.92	**-0.04**	54%
CIFAR-100	ResNet-56	NS [Liu et al., 2017] (Our-impl.)	72.49	71.40	1.09	24%
		Ours	72.49	72.46	**0.06**	25%
	VGG-16	NS [Liu et al., 2017] (Our-impl.)	73.83	74.20	-0.37	38%
		COP [Wang et al., 2019]	72.59	71.77	0.82	43%
		Ours	73.83	74.25	**-0.42**	43%

convolution layer in each residual block is padded with zeros to align with the input to the block and the details on pruning ResNet-50 are shown in Appendix C.

Table 2: Results on ImageNet. Best results are bolded.

Model	Approach	Baseline	Pruned		
		Acc. (%)	Acc. (%)	Acc. Drop (%)	FLOPs Reduction
ResNet-50	NS [Liu et al., 2017] (Our-impl.)	76.15	74.88	1.27	53%
	SSS [Huang and Wang, 2018]	76.12	71.82	4.30	43%
	DCP [Zhuang et al., 2018]	76.01	74.95	1.06	56%
	FPGM [He et al., 2019]	76.15	74.13	2.02	53%
	CCP [Peng et al., 2019]	76.15	75.21	0.94	54%
	MetaPruning [Liu et al., 2019b]	76.6	75.4	1.2	50%
	SFP [He et al., 2018a]	76.15	62.14	14.01	42%
	PFP [Liebenwein et al., 2020]	76.13	75.21	0.92	30%
	AutoPruner [Luo and Wu, 2020]	76.15	74.76	1.39	49%
	Ours	76.15	75.63	**0.52**	54%
MobileNet v2	AMC [He et al., 2018b]	71.8	70.8	1.0	27%
	MetaPruning [Liu et al., 2019b]	72.0	71.2	0.8	27%
	DeepHoyer [Yang et al., 2020] (Our-impl.)	72.0	71.7	0.3	25%
	Ours	72.0	71.8	**0.2**	28%

4.3 Experiment Results

We compare the performances of different pruning methods in terms of the accuracy and FLOPs reduction. For each method, we report its baseline model accuracy (Baseline Acc.), its fine-tuned accuracy after pruning (Pruned Acc.), and the accuracy drop (Acc. Drop) between baseline and pruned accuracy. The FLOPs are reported as the pruning ratio, i.e. the ratio of the FLOPs reduced by pruning to the FLOPs of the baseline model.

The CIFAR dataset is a relatively small dataset for image classification. It is the most widely used dataset to compare neural network pruning methods. The experiment results on CIFAR datasets are shown in Table 1. Note that a negative accuracy drop in Table 1 means that the pruned model gets better accuracy than its unpruned baseline model. On CIFAR-10, ResNet-56 task, our method

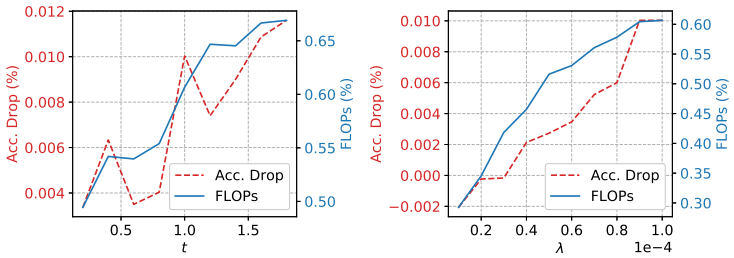


Figure 3: The effect of the hyper-parameters t and λ on the reduced FLOPs and accuracy drop for polarization pruning. When we draw the figure of t, the value of λ is fixed to 1e-4. When we draw the figure of λ, the value of t is fixed to 1.0. We conduct each experiment for 3 times and report the mean value of the accuracy drop and the FLOPs.

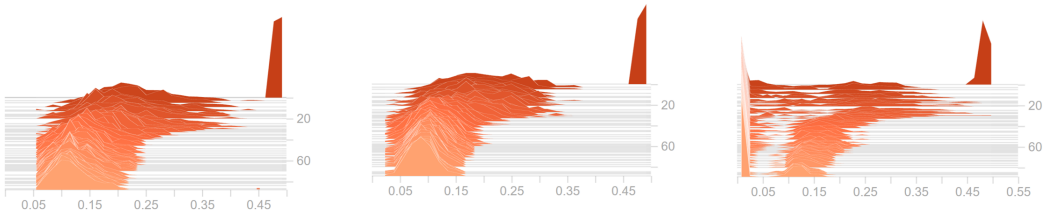
achieves the smallest accuracy drop (-0.03%) and the best pruned accuracy while pruning 47% FLOPs, which is much better than other methods. On CIFAR-10, VGG-16 task, our method also achieves the smallest accuracy drop and the best pruned accuracy, with FLOPs reduction similar to NS. On the CIFAR-100 dataset, few previous works on channel pruning have experimented and reported their results. So we only compare with NS and COP on CIFAR-100. And our method still achieves the best results as shown in Table 1.

The ImageNet dataset is a much larger dataset compared to CIFAR. The experiment results on ImageNet dataset are shown in Table 2. On ImageNet, ResNet-50 task, our method achieves the smallest accuracy drop and the best pruned accuracy with a FLOPs very close to other methods. MobileNet V2 is optimized for computation efficiency on mobile devices and contains less redundant computations. Therefore, it is more difficult to prune compared to ResNet-50. On ImageNet, MobileNet task, our method still has the smallest accuracy drop with similar FLOPs reduction to other methods. In summary, our method achieves state-of-the-art channel pruning results on both CIFAR and ImageNet datasets.

4.4 Analysis

The Effect of Hyper-parameters In Section 3.3, we point out that there is no theory to quantitatively describe how the number of reduced FLOPs depends on the hyper-parameters λ in Equation (1) and t in Equation (2). As shown in Figure 3, we empirically studied the effect of the hyper-parameters t and λ on the number of reduced FLOPs as well as the accuracy drop between baseline and the pruned model. Figure 3 shows that when t gets larger, the reduced FLOPs also gets larger. This is reasonable because according to Theorem 3.2, when t gets larger, polarization regularizer will push more scaling factors to 0, and thus more neurons/FLOPs will be pruned. At the same time, the overall trend of the accuracy drop is to gets larger, but with more fluctuations in the curve. This shows that the accuracy drop does not increase monotonically with respect to t. A larger t, and thus more FLOPs reduction, does not always lead to a larger accuracy drop. When λ gets larger, both of the FLOPs reduction and accuracy drop get larger monotonically. This is also reasonable because λ is the overall weight for polarization regularizer in Equation (1). As λ gets larger, polarization regularizer has more influence in training, causing more neurons being pruned and larger accuracy drop.

Visualizing the Effects of Regularizers in Pruning The effect of L1 and polarization regularizers can be visualized through the distribution/histogram of scaling factors. Although Figure 1 already shows an example, we visualize them in more details in Figure 4. Figure 4 shows the evolution of the distributions of scaling factors in the training process. In Figure 4, subfigures (a, b, c) correspond to the training process of baseline ResNet-50, pruning ResNet-50 using L1 regularizer, and pruning ResNet-50 using polarization regularizer on the ImageNet dataset respectively. In subfigure (c), we can clearly see that polarization regularizer gradually pushes the scaling factors to two clusters, with one cluster located at 0, and the other cluster located at a larger value. There is a clear margin between these two clusters. On the other hand, subfigure (b) shows that L1 regularizer pushes the scaling factors to one cluster located near 0.09. Visualization in this subsection verified our conclusion in Section 3.2 that polarization regularizer has the effect of pushing scaling factors to different poles, thus making the pruned and preserved neurons more separable.



(a) Baseline ResNet-50.

(b) Pruning ResNet-50 using L1 regularizer.

(c) Pruning ResNet-50 using polarization regularizer.

Figure 4: The histograms of scaling factors for neurons in a layer of ResNet-50 during the training process. The width, height, and depth of a histogram corresponds to the value of scaling factors, the number of scaling factors, and the training epochs.

Experiments on Wider Spectrum of FLOPs Reduction Previous works on channel pruning usually experiments with a FLOPs reduction no more than 50%. A larger FLOPs reduction may not favor a regularization-based pruning method like ours, due to the huge regularization imposed in the training loss. So we add experiments on larger FLOPs reduction. We compare with a simple channel pruning method that is not based on regularization: the Uniform Channel Scaling (UCS) method, which uses a width-multiplier to uniformly increases or decreases the channel counts across all layers with the same proportion. We also compare with two regularization based methods: NS and DeepHoyer. The original DeepHoyer uses pre-trained models before pruning with the regularizer, which requires extra pre-training epochs before pruning. For fair comparisons, we run all methods using the same experimental setup in our paper, which means exactly the same number of training epochs are used for all methods. The results are in Table 3, which shows that our method performs consistently better under a wide spectrum of FLOPs reductions.

Table 3: Results on large FLOPs reduction.

Dataset / Model	Approach	Baseline	Pruned		
		Acc. (%)	Acc. (%)	Acc. Drop (%)	FLOPs Reduction
CIFAR10 / ResNet56	UCS	93.80	92.25	1.55	70%
	NS [Liu et al., 2017] (Our-impl.)	93.80	91.20	2.60	68%
	DeepHoyer [Yang et al., 2020]	93.80	91.26	2.54	71%
	Ours	93.80	92.63	**1.17**	71%
ImageNet / ResNet50	UCS	76.15	73.78	2.37	70%
	NS [Liu et al., 2017] (Our-impl.)	76.15	70.61	5.54	70%
	Ours	76.15	74.15	**2.00**	70%
ImageNet / MobileNetv2	UCS	72.0	63.71	8.29	61%
	NS [Liu et al., 2017] (Our-impl.)	72.0	65.65	6.35	56%
	Ours	72.0	67.49	**4.51**	57%

5 Conclusion

In this paper, we propose a novel polarization regularizer and perform structured pruning by applying it on the scaling factors of neurons. Different from existing regularizers which push parameters to one pole (usually 0), polarization regularizer pushes parameters to two poles. We theoretically analyzed the properties of our polarization regularizer and empirically validate our analysis by examining the distribution of scaling factors in pruning. Extensive experiments on image classification tasks show that polarization regularizer produces the state-of-the-art pruning results, much better than the commonly used L1 regularizer. Our method is easy to implement and computationally efficient, requiring only one pass of network "pruning+fine-tuning". The polarization regularizer can be conveniently added to neural network training, suppressing some neurons and activating others simultaneously. In the future, we will find more applications where the polarization effect is needed.

6 Broader Impact

This work has the following potential positive impact in the society: because our work saves the computational cost of neural network inference, we can apply it to online inference services to reduce energy consumption, which is beneficial for environmental protection. We have not noticed any obvious negative consequences of our work.

Acknowledgments and Disclosure of Funding

This work is supported by Alibaba Group. The authors would like to than the Search and Recommendation Division of Alibaba Group for support of this work. The authors are also grateful to their colleagues in Alibaba who helped to provide the computing resources: Tao Lan, Pengfei Fan, Xiaochuan Tang, and Xiaowei Lu.

References

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019a.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 598–605. Morgan Kaufmann, 1989.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1135–1143, 2015.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 3592–3602, 2019.

Xiaohan Ding, Guiguang Ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, and Ji Liu. Global sparse momentum SGD for pruning very deep neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 6379–6391, 2019.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2074–2082, 2016.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2755–2763. IEEE Computer Society, 2017. doi: 10.1109/ICCV.2017.298.

Jianbo Ye, Xin Lu, Zhe Lin, and James Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: efficient inference engine on compressed deep neural network. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, pages 243–254. IEEE Computer Society, 2016. doi: 10.1109/ISCA.2016.30.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*, volume 11220 of *Lecture Notes in Computer Science*, pages 317–334. Springer, 2018. doi: 10.1007/978-3-030-01270-0_19.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2234–2240. International Joint Conferences on Artificial Intelligence Organization, 7 2018a. doi: 10.24963/ijcai.2018/309.

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

Hao Zhou, Jose M. Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 662–677, 2016. doi: 10.1007/978-3-319-46493-0_40.

Jose M. Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2262–2270, 2016.

Vadim Lebedev and Victor S. Lempitsky. Fast convnets using group-wise brain damage. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2554–2564. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.280.

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

Huanrui Yang, Wei Wen, and Hai Li. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. Collaborative channel pruning for deep networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5113–5122. PMLR, 2019.

Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognition*, page 107461, 2020. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2020.107461.

Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jin-Hui Zhu. Discrimination-aware channel pruning for deep neural networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 883–894, 2018.

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 *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*, pages 815–832, 2018b. doi: 10.1007/978-3-030-01234-2_48.

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 *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 3295–3304. IEEE, 2019b. doi: 10.1109/ICCV.2019.00339.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115 (3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, and Y. Yang. Asymptotic soft filter pruning for deep convolutional neural networks. *IEEE Transactions on Cybernetics*, pages 1–11, 2019. ISSN 2168-2275. doi: 10.1109/TCYB.2019.2933477.

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

Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*, 2016.

Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Appendices

A Proofs

This part contains the proofs of Lemma 3.1 and Theorem 3.2. For notational convenience, we use $f(\cdot)$ to substitute the $R_s(\cdot)$, and use $\boldsymbol{x}$ to substitute the $\boldsymbol{\gamma}$, in Equations (2) and (3) of Section 3.2. We also restate Lemma 3.1 and Theorem 3.2 using the new notations here, so that this part can be self-contained. Under the new notations here, Equation (2) in Section 3.2 becomes:

$$f(\boldsymbol{x}) = t\|\boldsymbol{x}\|_1 - \|\boldsymbol{x} - \bar{x}\mathbf{1}_n\|_1 = \sum_{i=1}^{n} t|x_i| - |x_i - \bar{x}|, (t \in \mathbb{R}, x_i \in [0, a]) \tag{5}$$

And Lemma 3.1 becomes:

Lemma A.1. *$f(\boldsymbol{x})$ defined in (5) is concave.*

Proof. For any $0 < \theta < 1$ and $x_i, y_i \in [0, a]$, we have:

$$\begin{aligned} &|\theta x_i + (1-\theta)y_i - [\theta\bar{x} + (1-\theta)\bar{y}]| \\ =&|\theta(x_i - \bar{x}) + (1-\theta)(y_i - \bar{y})| \\ \leq&\theta|x_i - \bar{x}| + (1-\theta)|y_i - \bar{y}| \end{aligned} \tag{6}$$

Using $t(\theta x_i + (1-\theta)y_i)$ to subtract both sides of inequality (6), we have:

$$\begin{aligned} &t(\theta x_i + (1-\theta)y_i) - |\theta x_i + (1-\theta)y_i - [\theta\bar{x} + (1-\theta)\bar{y}]| \\ \geq&\theta\left(tx_i - |x_i - \bar{x}|\right) + (1-\theta)\left(ty_i - |y_i - \bar{y}|\right) \end{aligned}$$

Therefore,

$$\begin{aligned} &\sum_{i=1}^{n} t(\theta x_i + (1-\theta)y_i) - |\theta x_i + (1-\theta)y_i - [\theta\bar{x} + (1-\theta)\bar{y}]| \\ \geq&\theta\sum_{i=1}^{n} tx_i - |x_i - \bar{x}| + (1-\theta)\sum_{i=1}^{n} ty_i - |y_i - \bar{y}| \end{aligned}$$

Note that when $x_i, y_i \in [0, a]$,

$$\begin{aligned} f(\boldsymbol{x}) &= \sum_{i=1}^{n} tx_i - |x_i - \bar{x}| \\ f(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) &= \sum_{i=1}^{n} t(\theta x_i + (1-\theta)y_i) - |\theta x_i + (1-\theta)y_i - [\theta\bar{x} + (1-\theta)\bar{y}]| \end{aligned}$$

So we have

$$f(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) \geq \theta f(\boldsymbol{x}) + (1-\theta)f(\boldsymbol{y})$$

which shows that $f(\boldsymbol{x})$ is concave on $\boldsymbol{x} \in [0, a]^n$.

□

Under the new notations here, Equation (3) in Subection 3.2 becomes:

$$\min_{\boldsymbol{x}\in[0,a]^n} f(\boldsymbol{x}) = \sum_{i=1}^{n} t|x_i| - |x_i - \bar{x}|, (t \in \mathbb{R}) \tag{7}$$

Given k indices: $1 \leq i_1, \cdots, i_k \leq n$, and k values: $v_1, \cdots, v_k \in \{0, a\}$, we define the notation:

$$\mathbb{F}_{i_{1,\cdots,k}, v_{1,\cdots,k}} = \{\boldsymbol{x} \in [0, a]^n | x_{i_1} = v_1, \cdots, x_{i_k} = v_k\} \tag{8}$$

Then we prove the following lemma:

Lemma A.2. *The optimal value of (7) is attained on the vertices of* $[0,a]^n$.

Proof. $[0,a]^n$ is an n-dimensional cube, and its boundary consists of $2n$ facets:

$$\begin{aligned}\partial[0,a]^n &= \bigcup_{i=1}^{n} (\{\boldsymbol{x} \in [0,a]^n | x_i = 0\} \cup \{\boldsymbol{x} \in [0,a]^n | x_i = a\}) \\ &= \bigcup_{i=1}^{n} (\mathbb{F}_{i,0} \cup \mathbb{F}_{i,a})\end{aligned}$$

where the notation of (8) is used. Obviously, $\mathbb{F}_{i,v} (v = 0 \text{ or } a)$ is a facet of $[0,a]^n$. Note that $\mathbb{F}_{i,v} = \{\boldsymbol{x} | x_i = v, \forall j \neq i, x_j \in [0,a]\}$. So $\mathbb{F}_{i,v}$ is a $n-1$ dimensional cube on the dimensions $1 \leq j \leq n, j \neq i$. Actually, $\mathbb{F}_{i_{1,\cdots,k}, v_{1,\cdots,k}}$ is a $n-k$ dimensional cube on dimensions $1 \leq j \leq n, j \notin \{i_{1,\cdots,k}\}$.

First step, we prove that the optimal value of (7) is attained on one facet of $[0,a]^n$. Assume $\boldsymbol{x}^* \in [0,a]^n$ is an optimal solution of (7). Because $[0,a]^n$ is a bounded and closed convex set on $\mathbb{R}^n$, then if $\boldsymbol{x}^* \notin \partial[0,a]^n$, there exists a line segment in $[0,a]^n$ passing through $\boldsymbol{x}^*$ and intersects with the boundary of $[0,a]^n$ on two points: $\boldsymbol{x}_1^{(n)}, \boldsymbol{x}_2^{(n)}$, i.e. $\exists \boldsymbol{x}_1^{(n)}, \boldsymbol{x}_2^{(n)} \in \partial[0,a]^n$, and $\exists \theta \in (0,1)$, such that $\boldsymbol{x}^* = \theta \boldsymbol{x}_1^{(n)} + (1-\theta)\boldsymbol{x}_2^{(n)}$. According to Lemma A.1, $f(\boldsymbol{x})$ is concave on $[0,a]^n$. Therefore $f(\boldsymbol{x}^*) \geq \theta f(\boldsymbol{x}_1^{(n)}) + (1-\theta) f(\boldsymbol{x}_2^{(n)}) \geq \min\{f(\boldsymbol{x}_1^{(n)}), f(\boldsymbol{x}_2^{(n)})\}$. Without loss of generality, suppose $f(\boldsymbol{x}_1^{(n)}) \leq f(\boldsymbol{x}_2^{(n)})$, then $f(\boldsymbol{x}^*) \geq f(\boldsymbol{x}_1^{(n)})$, which means $\boldsymbol{x}_1^{(n)}$ is also an optimal solution. Because $\boldsymbol{x}_1^{(n)} \in \partial[0,a]^n$, $\boldsymbol{x}_1^{(n)}$ must lie on one facet of $[0,a]^n$. Let $\mathbb{F}_{i_1,v_1}$ denote the facet that $\boldsymbol{x}_1^{(n)}$ lies on.

Second step, note that $\mathbb{F}_{i_1,v_1}$ is an $n-1$ dimensional cube, $\boldsymbol{x}_1^{(n)} \in \mathbb{F}_{i_1,v_1}$ is an optimal solution of (7), and $f(\boldsymbol{x})$ is concave on $\mathbb{F}_{i_1,v_1}$. For the same reasons as in the first step, the optimal value of (7) is attained on one facet of $\mathbb{F}_{i_1,v_1}$. Without loss of generality, suppose $\mathbb{F}_{i_{1,2},v_{1,2}}$ is the facet that the optimal solution lies on. Obviously, $\mathbb{F}_{i_{1,2},v_{1,2}}$ is an $n-2$ dimensional cube.

We can iteratively apply the same reasoning as the 2 steps above until we reach the n-th step, in which case the optimal solution $\boldsymbol{y}^* = (y_1^*, y_2^*, \cdots, y_n^*)$ lies on a 1-dimensional cube $\mathbb{F}_{i_{1,\cdots,n-1}, v_{1,\cdots,n-1}}$. Let $\{j\} = \{1,\cdots,n\} \setminus \{i_{1,\cdots,n-1}\}$, then $y_j^* \in [0,a]$, and $\forall i \neq j (1 \leq i \leq n), y_i^* \in \{0,a\}$. Suppose $y_j^* = \xi a, (0 \leq \xi \leq 1)$, and let $\boldsymbol{y}_1 = (y_1^*, \cdots, y_{j-1}^*, a, y_{j+1}^*, \cdots, y_n^*)$ and $\boldsymbol{y}_2 = (y_1^*, \cdots, y_{j-1}^*, 0, y_{j+1}^*, \cdots, y_n^*)$, then both $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are vertices of $[0,a]^n$, and $\boldsymbol{y}^* = \xi \boldsymbol{y}_1 + (1-\xi)\boldsymbol{y}_2$. Because $f(\boldsymbol{x})$ is concave, we have $f(\boldsymbol{y}^*) \geq \xi f(\boldsymbol{y}_1) + (1-\xi) f(\boldsymbol{y}_2) \geq \min\{f(\boldsymbol{y}_1), f(\boldsymbol{y}_2)\}$, i.e. $f(\boldsymbol{y}^*) \geq f(\boldsymbol{y}_1)$ or $f(\boldsymbol{y}^*) \geq f(\boldsymbol{y}_2)$. Because both $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are vertices of $[0,a]^n$, this shows that the optimal value is attained on vertices of $[0,a]^n$. □

Under the new notations here, Theorem 3.2 in Subection 3.2 becomes:

Theorem A.3. *A class of optimal solutions of (7) is that either* $\lfloor n\rho \rfloor$ *or* $\lfloor n\rho \rfloor + 1$ *number of* $x_i, (1 \leq i \leq n)$ *are a, and the rest are 0, where:*

$$\rho = \begin{cases} -t/4 + 1/2, & -2 \leq t \leq 2 \\ 0, & t > 2 \\ 1, & t < -2 \end{cases} \tag{9}$$

Proof. By Lemma A.2, the optimal solution of (7) can be found on the vertices of $[0,a]^n$: $\{\boldsymbol{x} | x_i \in \{0,a\}, 1 \leq i \leq n\}$. So (7) is solved by:

$$\min_{x_i \in \{0,a\}} f(\boldsymbol{x}) = \sum_{i=1}^{n} t|x_i| - |x_i - \bar{x}| \tag{10}$$

Suppose k variables of $x_i, (1 \leq i \leq n)$ equal a, and the rest $n-k$ variables equal 0. Then

$$\begin{aligned} f(\boldsymbol{x}) &= a\left(tk - \left[k|1 - \frac{k}{n}| + (n-k)|0 - \frac{k}{n}|\right]\right) \\ &= a\left(k(t - 1 + \frac{k}{n}) - (n-k)\frac{k}{n}\right) \end{aligned}$$

This shows that k determines the value of $f(\boldsymbol{x})$, which is permutation invariant. And because $a > 0$, solving (10) is equivalent to solving the following problem:

$$\min_{1\leq k\leq n} g(k) = k(t-1+\frac{k}{n}) - (n-k)\frac{k}{n} \tag{11}$$

Let $\rho = k/n$, and define

$$\begin{aligned} h(\rho) = \frac{g(k)}{n} &= \rho(t-1+\rho) - (1-\rho)\rho \\ &= 2\rho^2 + (t-2)\rho \end{aligned}$$

Because k can only be integers, the values of ρ are discrete. We first relax this discreteness constraint and assume ρ is a continuous variable in $[0, 1]$. Then (11) is simplified to:

$$\min_{0\leq\rho\leq 1} h(\rho) = 2\rho^2 + (t-2)\rho \tag{12}$$

Solving (12), we get the optimal solution:

$$\rho^* = \begin{cases} -t/4 + 1/2, & -2 \leq t \leq 2 \\ 0, & t > 2 \\ 1, & t < -2 \end{cases}$$

Now we consider the discreteness constraint. When $t > 2$, the optimal solution of (11) is 0. When $t < -2$, the optimal solution of (11) is n. When $t \in [-2, 2]$, $\rho^* \in [0, 1]$. If $\rho^* = m/n$, where $m \in \{0, 1, \cdots, n\}$, then the optimal solution of (11) is m. Otherwise, ρ^* must lie in some interval $(m/n, (m+1)/n)$, where $m \in \{0, 1, \cdots, n-1\}$. Because $\rho = \rho^*$ is the symmetric axis of the quadratic function (12), the minimal value of (12) on the discrete points $\{0, 1/n, 2/n, \cdots, 1\}$ must be attained either on m/n or on $(m+1)/n$. So the optimal solution of (11) is either m or $m+1$. □

B Detailed Parameters in Training

The detailed parameters in the training and pruning stages of our method are listed in Table 4. We use the most commonly used batch size, training epochs and learning rate decay scheme as in [Zagoruyko and Komodakis, 2016, Liu et al., 2017]. For example, in the training stage on ImageNet, the learning rate decay scheme "0.1,0.01,0.001@30,60" means that the learning rate in the first 30 epochs is 0.1, in epochs 31 to 60 is 0.01, and in epochs after 60 is 0.001. And "0.05 cosine lr decay" means using cosine learning rate decay strategy with the initial learning rate 0.05.

Table 4: Training hyper-parameters for sparsity training and fine-tuning. Learning rate denotes the learning rates and corresponding decay milestones.

Dataset	Model	Batch size	Training		Fine-tuning	
			Epoch	Learning rate	Epoch	Learning rate
ImageNet	ResNet	512	120	$\{0.1, 0.01, 0.001\}$ @{40,80}	128	0.05 cosine lr decay [Loshchilov and Hutter, 2017]
ImageNet	MobileNet v2	1024	256	0.4 cosine lr decay with warmup	256	0.05 cosine lr decay
CIFAR	ResNet & VGG	128	200	$\{0.01, 0.1, 0.02, 0.004, 0.0008\}$ @{1,60,120,160}	200	$\{0.001, 0.0005, 0.00025, 0.0001\}$ @{20,60,150}

C Details on Pruning ResNet-50

The main building block of ResNet-50 is the bottleneck block [He et al., 2016], as shown in Figure 5. The polarization regularizer is applied to the scale factors of the first two BN layers in each bottleneck block, i.e. the "`bn1`" and "`bn2`" in Figure 5. In the baseline ResNet-50 model, many scaling factors in the "`bn3`" of a bottleneck block are already 0 or very close to 0. So we do not apply any extra regularization on "`bn3`", either in our polarization pruning method or in our implementation of the NS method [Liu et al., 2017]. We visualize the layer-wise distribution of scaling factors in Figure 6. Figure 6 compares the layer-wise distributions of scaling factors between the baseline ResNet-50 model and the model trained with our polarization regularizer on ImageNet dataset. Due to space limit, we only visualize the distributions in the "conv2" and "conv3" layers as defined in [He et al., 2016], where "conv2" and "conv3" contains 3 and 4 bottleneck blocks respectively. Figure 6 shows that our polarization regularizer also makes the scaling factors more separable in layer-level.

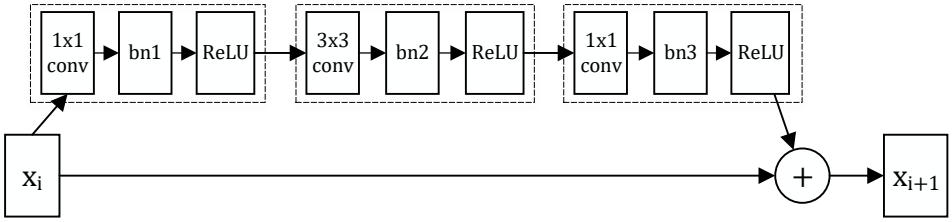


Figure 5: A "bottleneck" block in ResNet-50.

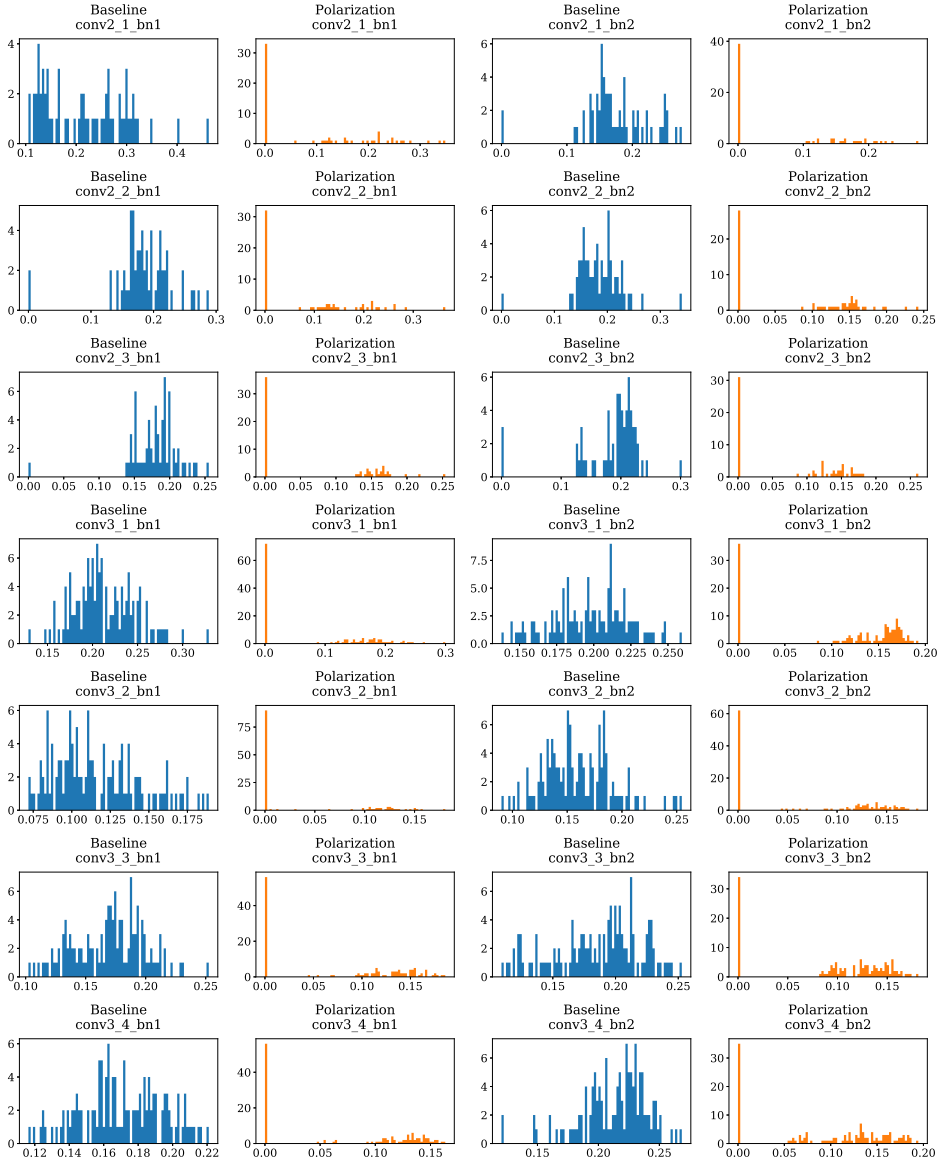


Figure 6: Comparison of the layer-wise scaling factor distributions between baseline ResNet-50 model and the model trained with our polarization regularizer on ImageNet dataset. The left two columns show the distributions of scaling factors in "bn1" of the bottleneck block. The right two columns show the distributions of scaling factors in "bn2" of the bottleneck block. The top 3 rows show scaling factor distributions in each bottleneck block of the "conv2" layer. The bottom 4 rows show scaling factor distributions in each bottleneck block of the "conv3" layer.