Pattern Recognition 140 (2023) 109508

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Sensitivity pruner: Filter-Level compression algorithm for deep neural networks☆

Suhan Guo[a,b], Bilan Lai[a,c], Suorong Yang[a,c], Jian Zhao[d,*], Furao Shen[a,b,**]

[a] State Key Laboratory for Novel Software Technology, Nanjing University, China
[b] School of Artificial Intelligence, Nanjing University, China
[c] Department of Computer Science and Technology, Nanjing University, China
[d] School of Electronic Science and Engineering, Nanjing University, China

ARTICLE INFO

Article history:
Received 26 October 2022
Revised 10 February 2023
Accepted 4 March 2023
Available online 10 March 2023

Keywords:
Filter pruning
Saliency-based pruning
End-to-end pruning framework
Sampling bias

ABSTRACT

As neural networks get deeper for better performance, the demand for deployable models on resource-constrained devices also grows. In this work, we propose eliminating less sensitive filters to compress models. The previous method evaluates neuron importance using the connection matrix gradient in a single shot. To mitigate the sampling bias, we integrate this measure into the previously proposed "pruning while fine-tuning" framework. Besides classification errors, we introduce the difference between the learned and the single-shot strategy as the second loss component with a self-adjustive hyper-parameter that balances the training goal between improving accuracy and pruning more filters. Our Sensitivity Pruner (SP) adapts the unstructured pruning saliency metric to structured pruning tasks and enables the strategy to be derived sequentially to accommodate the updating sparsity. Experimental results demonstrate that SP significantly reduces the computational cost and the pruned models give comparable or better performance on CIFAR10, CIFAR100, and ILSVRC-12 datasets.

1. Introduction

Deep neural networks have become the optimal solution for many tasks across industries, including weather forecasting, traffic control, and biomedical applications [1]. In recent years, the power of deep neural networks has become proportional to their depth and complexity, which means their deployment on resource-constrained devices, such as mobile phones, is very difficult. Hence, reducing the amount of calculation and memory load through model compression becomes necessary for model deployment. Currently, there are four types of model compression methods [2], including parameter pruning and sharing, low-rank factorization, transferred/compact convolutional filters, and knowledge distillation. Parameter pruning and sharing aim at removing redundant or less sensitive parameters. Low-rank factorization uses matrix decomposition to replace original filters with low-rank filters, which results in fewer overall parameters. Because decomposition is an operation of high computational complexity, the training process can be prolonged and power-demanding. Transferred/compact convolutional filters are meant to design transformation for layer output for compression purposes, which can achieve competitive results in wide networks but not narrow ones. Finally, to complete knowledge distillation, a pre-trained large network is used to train a smaller one, which can only be used on classification networks with softmax loss function, thus hampers applicability.

Parameter pruning, as a popular method in data compression, largely depends on its ability to distinguish redundant or less sensitive neurons from important ones. Common pruning methods, based on the granularity of the pruning unit, can be further categorized into structured pruning and unstructured pruning. Assuming equivalent compression rate and importance measure, it seems that neuron-level pruning, compared to filter-level pruning, should lead to better performance since selection at the lowest computation unit offers more possibilities to form a pruning strategy. Unfortunately, to enable the sparse model acceleration from unstructured pruning, special hardware, and library support are required. On the other hand, for structured pruning, though a larger pruning unit can potentially harm accuracy, we discover that filter-level pruning can achieve good performance with common library support and ordinary CPU/GPU devices.

☆ The code for Sensitivity Pruner (SP) is available at the GitHub repository: https://github.com/SuhanG17/Sensitivity_Pruner.git

* Corresponding author at: School of Electronic Science and Engineering, Nanjing University, Nanjing, China.
** Corresponding author at: School of Artificial Intelligence, Nanjing University, Nanjing, China.
E-mail addresses: shguo@smail.nju.edu.cn (S. Guo), mg1933032@smail.nju.edu.cn (B. Lai), sryang@smail.nju.edu.cn (S. Yang), jianzhao@nju.edu.cn (J. Zhao), frshen@nju.edu.cn (F. Shen).

https://doi.org/10.1016/j.patcog.2023.109508
0031-3203/

Previous works in filter-level pruning have proposed many ways to evaluate neuron importance. Some methods are as simple as measuring the absolute weight sum of each channel, and those that reach a certain magnitude are kept [3]. Recent methods are more intricate. Luo *et al.* [4] stated that if a subset of the input feature for the next layer produced the same output feature, corresponding filters for the complement set of this input feature subset were recognized as weak channels. Liu *et al.* [5] introduced reconstruction errors in intermediate layers to locate channels to be pruned. He *et al.* [6] considered the previously proposed norm-based criterion for filters has the following drawbacks: 1) deviation of filter norm distributions should be large enough for sufficient threshold search space, 2) filters with the minimum norm should be small that pruning them will not cause information loss. Thus, FPGM [6] defined filters near the geometric median to be the ones to keep as the saliency measure to avoid the aforementioned constraints. Inspired by AutoML, He *et al.* [7] searched for the optimal structure of the model instead of a connection to be pruned.

Among all previous works, two of them inspire ours the most: the AutoPruner [8] and the SNIP [9]. In AutoPruner, the training framework of "pruning while fine-tuning" is proved to promote pruning performances, and an adaptive compression ratio across layers is demonstrated to reduce training costs. Despite the success of AutoPruner, the drawback of their filter importance measure is that while applying pooling and convolution on feature maps may be able to collect information across batches as claimed, the relationship between such information and filter importance is not explicitly explained. In SNIP, however, the importance of each neuron is expressly defined as the gradient with respect to the connection mask (an indicator matrix of 0s and 1s, where 0 indicated removal of connection, 1 indicated keeping), which can be calculated in a single shot. In their theoretical definition, the entire dataset should be used to calculate the pruning strategy. In reality, however, it is usually infeasible to handle a dataset as a single batch. By randomly selecting a batch, as did in SNIP, the sampling bias will inevitably affect the final strategy, rendering a single shot inadequate. Therefore, our question is: **How can we use the defined neuron importance for the entire dataset to generate a bias-free pruning strategy**?

We integrate the sensitivity measure from SNIP into the "training while fine-tuning" framework to form a more powerful pruning strategy. Most methods concerning pruning pre-trained models are classification error based, which seek to minimize the error after pruning. Recent works, such as [4,5,8,10], introduced additional components to loss functions to regularize the compression process. In our work, we introduce the difference between the learned pruning strategy and the single-shot strategy as the second loss component. With a hyper-parameter λ to balance two loss terms, we can dynamically tweak the training goal between improving model accuracy and pruning more filters.

Our key advantages of Sensitivity Pruner are summarized as follows:

- We integrate the sensitivity measure from SNIP into the "training while fine-tuning" framework to form a more powerful pruning strategy by adapting the unstructured pruning measure from SNIP to allow filter-level compression. In practice, the sensitivity score can be easily computed as the gradient of the connection mask applied to the weight matrix. Independent of the model structure, the sensitivity score can be applied to most neural networks for pruning purposes.
- We mitigate the sampling bias in the single-shot influence score by introducing the difference between the learned pruning strategy and the single-shot strategy as the second loss component. Filter influence is measured on batched data, where a convolutional layer is used to discover the robust influence from the noise of the batch. The learning process is guided by the score provided by the influence measure.
- Our algorithm can dynamically tweak the training goal between improving model accuracy and pruning more filters. We add a self-adaptive hyper-parameter λ to balance between classification and pruning strategy loss. To accommodate this proposed loss, the pruning strategy is derived sequentially to enable the current layer to be pruned according to the latest network sparsity.
- Most pruned networks appear to deliver better results than the pre-trained network on small datasets and a good result on the big dataset. To demonstrate the impact of the sampling bias on the influence score, we empirically analyze the correlation between the learned pruning strategy and the generally correct single-shot strategy.

2. Related work

The CNN module has become the fundamental building block of many networks, and the convolution kernels are typically four-dimensional tensors. Observing redundancy in these tensors, tensor decomposition offers a promising compression solution. Some works assumed the best rank-K approximation exists in general and tried to find it by training a low-rank constrained CNN [11]. Others, refuting such an assumption, used nonlinear least squares to compute the CP-decomposition [12] and then applied discriminative fine-tuning for better accuracy.

Though lacking strong theoretical proof, empirical evidence has demonstrated that the translation invariant property and convolutional weight-sharing are beneficial to model performance [2]. The rationale behind transferred convolutional filters is based on the equivariant group theory [13], which states that the transformation of input before or after passing the network layer should be equivalent. Although the transformation itself may be different, the projection should stay the same. Therefore, many works have been focused on developing the right transformation for network compression. For example, because lower convolutional layers have redundant filters to extract both positive and negative phase information from an input signal, [14] defined the transformation to be a simple negation function, which can achieve $2\times$ compression rate on all layers.

In knowledge distillation, softened softmax is used for a small model to learn the class distribution output in a larger teacher model [2]. Romero *et al.* [15] proposed to replace wide and shallow networks with thin and deep networks for model compression, which impelled the student to mimic the full feature maps of the teacher. Luo *et al.* [16] treated neurons in the top hidden layer as knowledge, which preserved as much information but was more compact. With the function preserving transformations, Chen *et al.* [17] could instantaneously transfer the knowledge from the previous network to each new deeper or wider network.

In terms of parameter pruning and sharing, previous work has found that model pruning can not only reduce network complexity but also address overfitting issues, which can be further categorized into model quantization/binarization and parameter sharing.

Model quantization uses low bitwidth integers to represent the weights and activations [2]. Lower bit representations, such as 8-bit and 16-bit, are shown to be able to reduce memory usage and float-point operations with acceptable loss in classification accuracy [18]. In extreme cases of quantization, binary integer weights are used to train networks from scratch [19,20], which trade accuracy to gain significant speed-up. The search for the bitwidth of the best fit is crucial for model performance because discrete weights may deliver worse performance if bitwidth is too low. Also, in a discrete setting, the weights of neural networks become non-differentiable, which renders the training very difficult.

Pruning and sharing methods aim to remove redundant modules, including channels or kernels, to accelerate the run-time inference. Some works have explored ways to find compact models by training from scratch. Earlier research use the Hessian of the loss function to select the weak connections. More recent methods introduce sparsity constraints in the objective function as the l_0 and l_1-norm regularizers [21,22]. Other works focus more on pruning a pre-trained model. Earlier research develop magnitude-based weight selection based on a predetermined threshold [23,24], rendering irregular convolution and requiring designed hardware to realize computational savings. Recent methods, concentrate more on structured pruning and choose to measure the module importance of the network by proposing different metrics [3,6,9]. Furthermore, to improve pruning efficiency, reconstruction-based methods formulate the channel selection problem into the optimization of the reconstruction error [3–5,25]. To further compress the pruned model, some methods seek to combine sharing with quantization. For instance, Han *et al.* [23] quantized the weights after removing redundancy and then used Huffman coding to encode the quantized weights.

Among studies of network pruning, two methods inspired us the most, namely, Autopruner [8] and SNIP [9]. In Autopruner, authors stacked up a framework where a pruning strategy is gradually developed during training. The framework also comes with several tricks to facilitate the training process, including scaled sigmoid to ensure binarization of pruning mask, "pseudo" pruning where they applied the 0–1 mask on the weight matrices using Hadamard product to update pruning strategy during training, gradient-based mask learned from the gradient of weight matrices and gathering information across batches by using convolutional layers and pooling layers. SNIP, on the other hand, mathematically defined neuron importance as the difference in loss if such a neuron were removed. By their definition, the aforementioned importance could be approximated using the derivative of the loss function with respect to the connection matrix, which is a 0–1 matrix indicating if a neuron is pruned or active. They claimed that the strategy only needed to be calculated once using a random batch of data. After fine-tuning, the sparse network should reach its prime in terms of Top-1 Accuracy.

3. Sensitivity pruner

In this section, we propose our sensitivity-based end-to-end pruning method: Sensitivity Pruner (SP). We will give a comprehensive introduction to the SP pipeline as well as several important implementation details.

The pipeline for the pruning strategy generation is shown below. Two models with the same structure are used, demonstrated by the upper and the middle of Fig. 2. One of them is used to generate the influence matrix (the upper part), and the other (the middle part) is to generate and apply the "pseudo" strategy using element-wise multiplication and then train for accuracy. After the pruning strategy is finalized, we implement the pruning strategy by removing the pruned filters accordingly and fine-tuning this smaller model for better accuracy as shown at the bottom of Fig. 2.

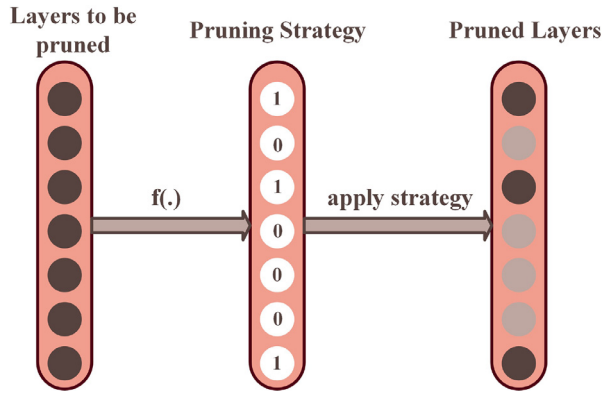

Fig. 1. Parameter Pruning, a designed metric function measures neuron importance and upon which, a pruning strategy is generated, denoted by $f(.)$. This strategy guides the pruning process of the neurons.

To generate the pruning strategy, in one of the networks, the indicator matrix, initiated with all 1s, is applied to all layers to be pruned. The gradient of this matrix is regarded as the influence matrix, denoted as "$\mathbf{Inf}_l$" in Fig. 2. The influence matrix is then summed over the output channel so each channel possesses an importance score. Then, the importance score for all channels to be pruned undergoes a sorting process, and a threshold score defined by a preset compression rate is selected. For the current layer, channels with scores larger than the threshold will be retained; otherwise, they will be pruned. This indicator vector is regarded as the single-shot strategy, which will be the ground truth for the current epoch.

On the other hand, the learned strategy is generated from the normalized influence matrix. A convolutional layer extracts features from this matrix for all batches. The output of this convolutional layer is a tensor with a width of channel numbers, and each element indicates channel importance. Then, this importance vector is binarized by a scaled sigmoid to produce masks as the temporary learned strategy. This strategy is used in another network of the same structure by applying this mask to layers of interest as shown in Fig. 3.

The learned strategy will be updated every 20 training steps with a function to push temporary strategy to 0s and 1s. After training, the learned strategy will guide the actual pruning by

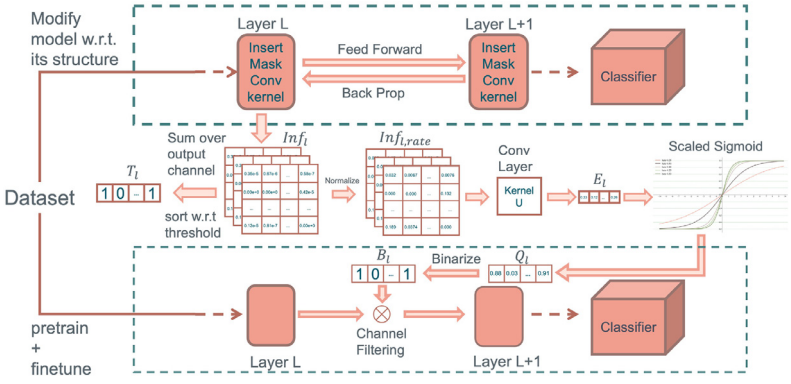

Fig. 2. Sensitivity Pruner Pipeline.

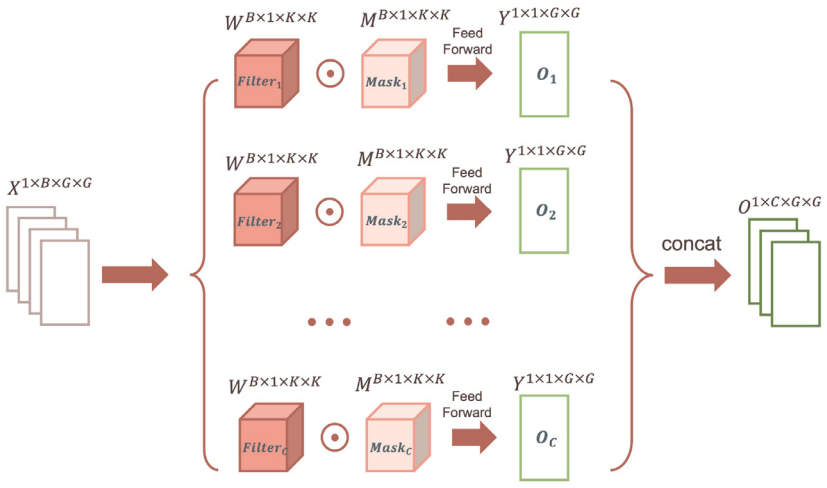

Fig. 3. Apply the temporary learned strategy to layers of interest as "pseudo pruning".

removing the output channels of layers at value 0 and retaining the rest.

Next, we will go into detail about the proposed method. Our method consists of four major parts: sampling bias, sensitivity definition, binarization, and loss function, which will be covered by Section 3.2 ∼ 3.5, respectively.

3.1. Preliminary

First, we define three matrices in layer l among an N layer CNN network: $\boldsymbol{W}_l \in \mathbb{R}^{B\times C\times K\times K}$, $\boldsymbol{M}_l \in \{0,1\}^{B\times C\times K\times K}$, and $\boldsymbol{W}_l' \in \mathbb{R}^{B\times C\times K\times K}$, where B, C, K denotes the number of input channels, number of output channels and kernel size respectively. In the latter paragraphs, b, c, j, and k will index the weight matrix as input channel, output channel, kernel map rows, and kernel map columns respectively. Applying Hadamard product between weight matrix $\boldsymbol{W}_l$ and connection matrix $\boldsymbol{M}_l$, we can have pruned weight matrix $\boldsymbol{W}_l' = \boldsymbol{W}_l \odot \boldsymbol{M}_l$.

Second, for a dataset $\mathcal{D}$, the loss function of this layer for a pre-trained model is defined as:

$$L(\boldsymbol{W}_l \odot \boldsymbol{M}_l; \mathcal{D}), \quad \text{where} \quad \boldsymbol{M}_l = \{0,1\}^{B\times C\times K\times K}. \tag{1}$$

3.2. Sampling bias

Notice that in SNIP [9], authors defined the pruning problem as a constrained optimization problem, and the neuron importance was approximated using the change in loss before and after the subtraction of an infinitesimal change. Accordingly, to evaluate the neuron importance, the entire dataset should be used, where L indicates loss over all batches. In practice, only a single batch was selected randomly, which made their formula subject to sampling bias and the pruning strategy can be misguided by the noise in the selected batch. To mitigate sampling bias and take the RAM of GPU into consideration, a training process iterating over the entire dataset seems to be the optimal solution. Hence, we re-define the loss function as the average of sample losses for batch $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$ as:

$$\begin{aligned} &\hat{L}(\boldsymbol{W}_l \odot \boldsymbol{M}_l; \{\boldsymbol{x}_i, \boldsymbol{y}_i\}), \quad \text{where} \quad \boldsymbol{M}_l = \{0,1\}^{B\times C\times K\times K}, \\ &L(\boldsymbol{W}_l \odot \boldsymbol{M}_l; \mathcal{D}) \quad = \frac{1}{n}\sum_{i=1}^{n} \hat{L}(\boldsymbol{W}_l \odot \boldsymbol{M}_l; \{\boldsymbol{x}_i, \boldsymbol{y}_i\}). \end{aligned} \tag{2}$$

3.3. Sensitivity definition

If we were to prune a connection in weight $\boldsymbol{W}_l$ at input channel P, output channel Q, and kernel map position (M, N), we can simply set the same position in connection matrix to zero, shown as:

$$\boldsymbol{W}'_{l,(b,c,j,k)} = \boldsymbol{W}_l \odot (\boldsymbol{1} - \boldsymbol{M}_{l,(b,c,j,k)}), \tag{3}$$

$$\boldsymbol{M}_{l,(b,c,j,k)} = \begin{cases} 1, & b = P, c = Q, j = M, k = N, \\ 0, & \text{otherwise.} \end{cases}$$

The change in loss before and after pruning a given connection is defined as the sensitivity score of a given neuron. $\boldsymbol{M}_{l,(P,Q,M,N)}$ is an indicator variable for neuron at layer l, input channel P, output channel Q, and kernel map position (M, N). Depending on whether this neuron is active or pruned, we have $m_{l,(P,Q,M,N)} = 1$ or $m_{l,(P,Q,M,N)} = 0$, respectively. In order to approximate the sensitivity as the rate of change in $\hat{L}$ with respect to the connection of interest, the binary constraint on the indicator variable $\boldsymbol{M}_{l,(P,Q,M,N)}$ is relaxed to be continuous. This derivative is denoted as $f_{l,(b,c,j,k)}(\boldsymbol{W}_l; \{\boldsymbol{x}_i, \boldsymbol{y}_i\})$ and its formula is demonstrated as:

$$\begin{aligned} \Delta\hat{L}_{l,(b,c,j,k)}(\boldsymbol{W}_l; \{\boldsymbol{x}_i, \boldsymbol{y}_i\}) &\approx f_{l,(b,c,j,k)}(\boldsymbol{W}_l; \{\boldsymbol{x}_i, \boldsymbol{y}_i\}) \\ &= \frac{\partial \hat{L}(\boldsymbol{W}_l \odot \boldsymbol{M}_l; \{\boldsymbol{x}_i, \boldsymbol{y}_i\})}{\partial m_{l,(b,c,j,k)}}\bigg|_{\boldsymbol{M}_l=\boldsymbol{1}} \\ &= \lim_{\alpha\to 0} \frac{\hat{L}(\boldsymbol{W}_l \odot \boldsymbol{M}_l; \{\boldsymbol{x}_i, \boldsymbol{y}_i\}) - \hat{L}(\boldsymbol{W}_l \odot (\boldsymbol{M}_l - \alpha\boldsymbol{M}_{l,(b,c,j,k)}); \{\boldsymbol{x}_i, \boldsymbol{y}_i\})}{\alpha}\bigg|_{\boldsymbol{M}_l=\boldsymbol{1}}. \end{aligned} \tag{4}$$

With the defined sensitivity measure, we can evaluate the neuron importance by taking the derivative with respect to the connection mask $\boldsymbol{M}_l$. The sensitivity of weight matrix $\boldsymbol{W}_l$, denoted as the influence matrix $\mathbf{Inf}_l$, can be obtained easily in one feed-forward and back-propagate iteration using auto-differentiation provided by common frameworks, such as PyTorch and TensorFlow. To apply the unstructured pruning saliency metric on a structured pruning task, we define the channel importance $\boldsymbol{S}_l \in \mathbb{R}^{1\times C}$ by summing the influence matrix over output channels, i.e.,

$$\boldsymbol{S}_l = \sum_{b=1}^{B}\sum_{i=1}^{K}\sum_{j=1}^{K} f_{l,(b,c,j,k)}(\mathbf{Inf}_l; \{\boldsymbol{x}_i, \boldsymbol{y}_i\}). \tag{5}$$

With the predetermined compression rate r, the number of layers to be pruned P, and the output channel C_l for each layer, we apply descending sort to keep channels of higher importance based on threshold t, where t is defined as:

$$t = \sum_{l=1}^{P} C_l \times r. \tag{6}$$

The descending sort across all channels is not always optimal. With experiments, we discover that global sort on networks generates good results on smaller datasets, such as CIFAR-10 and CIFAR-100. For ILSVRC-12, sorting applied to each layer of interest individually produces better results. Hence, besides the global sort shown above, we also have layer sort to find the threshold for each layer:

$$t_l = C_l \times r. \tag{7}$$

3.4. Binarization

So far, for layer l, a single-shot pruning strategy $\boldsymbol{T}_l \in \{0,1\}^{1\times C}$ can be deducted by keeping channels with channel importance $\boldsymbol{S}_l$ larger than t or t_l. On the other hand, the first step to obtain the learned pruning strategy $\boldsymbol{B}_l \in \{0,1\}^{1\times C}$ is to calculate strategy feature $\boldsymbol{E}_l \in \mathbb{R}^{1\times C}$ by applying convolutional kernel $\boldsymbol{U} \in \mathbb{R}^{C\times C\times 3\times 3}$ on the normalized influence matrix, which is summed over batch. Because the learned strategy should be decided by the layer and not a particular sample, we used the summation and the convolutional kernel to collect contributions from all samples. Then, the "pseudo" strategy $\boldsymbol{Q}_l \in \mathbb{R}^{1\times C}$ is calculated by binarize the strategy feature $\boldsymbol{E}_l$ with the sigmoid function, scaled with β. After training, the pruning strategy $\boldsymbol{B}_l$ is finalized by pushing $\boldsymbol{Q}_l$ to 0s and 1s, i.e.,

$$\boldsymbol{Q}_l = \text{sigmoid}\,(\beta \boldsymbol{E}_l), \tag{8}$$

$$\boldsymbol{B}_l = (\text{sign}\,(\boldsymbol{Q}_l - 0.5) + 1)/2.0. \tag{9}$$

During training, the pruning is implemented by applying $\boldsymbol{Q}_l$, not $\boldsymbol{B}_l$, to the weight matrix using the Hadamard product. The "pseudo" strategy allows the network to revive a neuron even if deemed less sensitive, so the strategy will not stop optimizing until convergence. Moreover, using "ReLU" as the activation function, a connection marked as "pruned" will be projected to 0 while the "active" connection will maintain its value. Thus, if we remove the pruned channels after training, the prediction will not be affected.

The hyper-parameter β is introduced to map the strategy feature $\boldsymbol{E}_l$ to 0s and 1s as much as possible. The β is an amplifier, and as it grows gradually, the output of the sigmoid function will become approximately binary. The increasing of β is crucial because the strategy is slowly finalized as the network training converges and β makes binarization happen at a proper training step. When β is small, the convolutional kernel $\boldsymbol{U}$ can be trained, but it will take longer for the strategy to converge. However, if β is large, values in $\boldsymbol{E}_l$ will be mapped to 0s and 1s automatically, meaning $\boldsymbol{U}$ can be downgraded to random selection. Therefore, we wish to increase β gradually throughout the training process to allow convolutional kernel $\boldsymbol{U}$ to be fully trained. The initiation for β varies if different datasets and model structures are prepared to be pruned. In the latter chapters, we will demonstrate the effect of different β initiations with experiments.

3.5. Loss function

The loss function is designed to guide the pruning strategy training by adding the difference between the single-shot strategy and the learned strategy to classification loss.

$$L = L_{\text{classification}} + \lambda \|\mathbf{Q}_l - \mathbf{T}_l\|_2^2. \tag{10}$$

We presume that the single-shot strategy is generally correct but susceptible to sampling bias. Hence, in the second term of equation (10), we think of the single-shot strategy as the teacher, and the "pseudo" strategy should learn from the teacher while remaining generalizable to the entire dataset. λ is used to balance between maintaining classification accuracy and pruning more connections as desired. $\frac{\boldsymbol{T}_l}{\|\boldsymbol{C}_l\|_0}$ denotes the ratio of connections intended to be kept and $\frac{\boldsymbol{B}_l}{\|\boldsymbol{C}_l\|_0}$ denotes the ratio of connection actually be kept. When the ratio kept is smaller than intended, we increase λ to encourage the method to prune more connections and set λ to 0 to focus on classification accuracy otherwise. Notice that because both the intended ratio and the actual ratio are calculated based on the network results, the actual compression rate can vary; thus, the network compression is adaptive.

$$\lambda = \begin{cases} 5.0 \times \left| \frac{\boldsymbol{T}_l}{\|\boldsymbol{C}_l\|_0} + \frac{\boldsymbol{B}_l}{\|\boldsymbol{C}_l\|_0} - 1 \right|, & 1 - \frac{\boldsymbol{B}_l}{\|\boldsymbol{C}_l\|_0} \geq \frac{\boldsymbol{T}_l}{\|\boldsymbol{C}_l\|_0}, \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

3.6. FLOPs Computation

Following the setting of [8], the FLOPs in convolutional layers are defined in equation (12) with H, W, C_{in}, C_{out} and K, denoting the height, the width, the input channel number, the output channel number, and the kernel size, respectively. The 1 in the equation below means the FLOPs in bias term.

$$FLOPs = 2HW\,(C_{in}K^2 + 1)C_{out}. \tag{12}$$

As we prune one output channel, the FLOPs drop is calculated with $2HW\,(C_{in}K^2 + 1)$. For networks with repeated block structures, such as ResNet-56 and MobileNetV2, pruning more channels in shallower blocks will result in more FLOPs drop because the height and width of the current feature map are bigger than those in deeper blocks. Let us take ResNet-56 as an example, if we prune one output channel away from the middle 3×3 kernel in the first block with 64 output channel and 64 input channels, we obtain the FLOPs drop of 3,618,944. However, for the same kernel size in the second block with 128 as input and output channel number, we only obtain the FLOPs drop of 1,807,904. Unless the network chooses to prune many more channels in deeper blocks, pruning shallow blocks seems more "cost-effective."

3.7. Algorithm summarization

The SP is recapitulated in the pseudocode 1. For each training step, the influence matrix is retrieved from one network, and the learned strategy and target strategy are both derived from this matrix, which will later be fed to the loss function with the prediction to encourage pruning for as many filters as intended. Two types of binarization are involved, the first type is the scaled sigmoid with β updated in each iteration and the second type is to discretize the learned strategy after it has been deemed final.

4. Experiments

We demonstrate the effectiveness of the proposed method by applying the Sensitivity Pruner to various architectures, including VGG-16 [26], ResNet-56 [27] and MobileNetV2 [28]. In VGG-16 and ResNet-56, the structure is slightly altered by adding the Batch-Normalization layer. We conduct experiments on image classification with dataset CIFAR-10 [29], CIFAR-100 [29] and ILSCVR-12 [30]. All implementations are based on PyTorch. The discussion of experiments is as follows: we evaluate our method on image classification in Section 4.3 $\sim$ 4.5; we conducted ablation studies on hyper-parameters and training tricks in Section 5.1 $\sim$ 5.3; and we illustrate our method can alleviate sampling bias in Section 5.4.

4.1. Compared methods

To investigate the effectiveness of the proposed methods, we compare our methods with several state-of-the-art channel pruning methods. On CIFAR-10, we compare Sensitivity Pruner with

Algorithm 1 Sensitivity Pipeline.

Require: Proposed Loss function L and classification loss L_c,
β initiation β_0 and β range (L, U)
Sensitivity Model $\mathbb{S}$ and Strategy application model $\mathbb{N}$
Ensure: Pruning strategy Mask $\boldsymbol{B}_l$
β-scale $\leftarrow$ LinearFunction(β_0, (L, U)) ▷ evenly space out β values on a linear scale
for each layer l **do**
  for each epoch **do**
    for each batch **do**
      $\mathbf{Inf}_l \leftarrow \text{Backprop}(L_c(\mathbb{S}(\boldsymbol{x}_i), \boldsymbol{y}_i))$ ▷ Obtain gradient as influence matrix
      $\boldsymbol{T}_l \leftarrow \text{SortingAndThresholding}(\mathbf{Inf}_l)$, ▷ Refer to Eq. 5,6,7
      $\boldsymbol{E}_l \leftarrow \boldsymbol{U}(\boldsymbol{T}_l)$ ▷ Derive strategy from all batches
      $\beta \leftarrow \text{BetaUpdate}(\beta\text{-scale})$ ▷ Update β for this sample
      $\boldsymbol{Q}_l \leftarrow \text{ScaledSigmoid}(\beta, \boldsymbol{E}_l)$
      $\hat{\boldsymbol{y}} \leftarrow \mathbb{N}(\boldsymbol{x}, \boldsymbol{Q}_l)$ ▷ Apply strategy
      $\delta \leftarrow L(\hat{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{Q}_l, \boldsymbol{T}_l)$ ▷ Refer to Eq. (10)
      $\mathbb{N} \leftarrow \text{BackpropAndUpdate}(\Delta)$ ▷ model weights updated
    end for
    $\boldsymbol{B}_l \leftarrow \text{Binarization}(\boldsymbol{Q}_l)$ ▷ Refer to Eq. (9)
  end for
  $\mathbb{N} \leftarrow \text{Prune}(\mathbb{N}, \boldsymbol{B}_l)$ ▷ Remove filters in layer l w.r.t. $\boldsymbol{B}_l$
end for

FPGM [6], NS [31], NSP [10] using VGG-16; CCP [32], AMC [7], CP [25], FPGM [6], SFP [33], ResRep [34] using ResNet-56. On CIFAR-100, we compare Sensitivity Pruner with NS [31], COP [35], NSP [10] using VGG-16; NS [31] and NSP [10] using ResNet-56. On ILSVRC-12, we compare Sensitivity Pruner with SFP [33], MetaPruning [36], ResRep [34], Adapt-DCP [5], C-SGD [37] and ABCPruner [38] using ResNet-56; MetaPruning [36], AMC [7], and DMCP [39] using MobileNetV2. We also compare our method with AutoPruner [8] and SNIP [9] for all tasks and datasets. Following previous publications [5,8,31], we measure the computational cost of the pruned models by the number of FLOPs.

4.2. Datasets and implementation details

Three datasets, including CIFAR-10, CIFAR-100, and ILSVRC-12, are used in image classification tasks to represent scenarios of small, medium, and large classification tasks. CIFAR-10 contains 50k training images and 10k testing images with 10 classes. CIFAR-100 has the same 50k training images and 10k testing images with 100 classes. ILSVRC-12 consists of 1.28 million training images and 50k testing images for 1000 classes.

Upon pre-trained models, we insert our sensitivity measuring layer on one model to gather the channel importance and generate a pruning strategy. Then, we apply this strategy to another model to complete "pseudo" pruning, which is trained with the aforementioned two-part loss.

We test our method on the three most popular network structures: VGG-16, ResNet-56, and MobileNetV2. In ResNet-56, the residual connection is added to tackle the vanishing gradient, which is implemented using the simple addition of an input feature map and output feature map for a certain block. To ensure such addition, the last layer of blocks with residual connection ahead is not pruned. In MobileNetV2, point-wise convolution is implemented using the "group" parameter in the PyTorch framework. In every Inverted Residuals block, only the importance of output channels of the first convolutional layer, which is the depth-wise convolution, is measured and pruned accordingly.

In ResNet-56, blocks with similar structures but different channel numbers are prevalent. Therefore, we further divide the model into multiple groups. Then, the model is pruned group by group: when we have determined the strategy for a group, the network is compressed with this strategy before it goes on to find out the strategy for the next group. In VGG-16, similar techniques are adopted but with inherent layers, not groups. In MobileNetV2, under the same concern, we chose to train depthwise separable convolution kernels block by block and treated the last classification layer as the last block. Global sorting is applied to ResNet-56 only on CIFAR-10 and CIFAR-100, and to VGG-16 on all three datasets. On ILSVRC-12, we apply layer sort to both ResNet-56 and MobileNetV2 for better results.

As we mentioned, the initiation of β is significant for model performance. We initiate β to be 1 for VGG-16 and ResNet-56, 0.05 for MobileNetv2. The β range denotes the approximate range for β during training, which is 100 for VGG-16 and ResNet-56, and 5 for MobileNetV2. Because we want the β to change dynamically for different layers, the β value may exceed the highest value set if the learned pruning strategy is not binary enough.

We use SGD optimization with momentum and weight decay set to 0.9 and 5e-4, and we initiate the learning rate to be 1e-3 for all datasets and all models. For VGG-16, with CIFAR-10, and CIFAR-100, we use a total of 3 epochs and a batch size of 12 for each layer, and with ILSVRC-12, we use a total of 1 epoch and a batch size of 24 for each layer. The learning rate is divided by 10 every 2 epochs per layer for CIFAR-10 and CIFAR-100. For ResNet-56, with CIFAR-10, and CIFAR-100, we use a total of 8 epochs and a batch size of 12 for each group, and with ILSVRC-12, we use a total of 4 epochs and a batch size of 48 for each group. The learning rate is divided by 10 every four epochs per group. For MobileNetv2, we use a total of 4 epochs and a batch size of 64 for each block. The learning rate is divided by 10 every two epochs.

After the model is compressed, we finetune the new model for 200 epochs, with a learning rate of 1e-4. SGD optimizers have momentum and weight decay set to 0.9 and 1e-4. The learning rate is initialized to 1e-4 and decreased by 10 at epochs 66, 132, and 198.

4.3. Comparisions on CIFAR-10

We apply Sensitivity Pruner on ResNet-56 and VGG-16 and compare the performance on CIFAR-10. The results are reported in Table 1. Compared with several state-of-the-art methods, our method achieves the best performance. For instance, NSP prunes VGG-16 to reduce 54.00% of the FLOPs with a 0.04% drop in the Top-1 accuracy. In contrast, our proposed Sensitivity Pruner achieves 57.00% of speed-up ratio with a 0.28% improvement in the Top-1 accuracy.

In terms of ResNet-56, one of the current best methods ResRep achieves zero loss in the Top-1 accuracy with a 52.91% drop in FLOPs. In Our model, we attain a 0.43% increase in Top-1 accuracy with a 53.00% drop in FLOPs. Compared with CCP, our method increases the model accuracy as much as they do but with a 1.00% more reduction in FLOPs. These results have shown the superior performance of our proposed method on small datasets, such as CIFAR-10. It is worth discussing that one of the state-of-the-art models, ResRep, is not among the best in using the ResNet-56 structure on CIFAR-10. We presume that our model is tailored to small datasets and ResRep is more suitable to larger datasets.

4.4. Comparisions on CIFAR-100

Sensitivity Pruner is also tested on CIFAR-100 with ResNet-56 and VGG-16. In Table 2, we report that our method performs well in either accuracy or speed-up ratio. In VGG-16, compared with other methods, though our method did not increase the Top-1 accuracy by the highest margin, we achieve the best FLOPs Drop ratio. In ResNet-56, we improve the Top-1 accuracy by 0.09% with

Table 1
Performance comparisons on CIFAR-10. Top-1 Acc. ↑ is the Top-1 Accuracy gap between the pruned model and the baseline model. * means for this method, the FLOPs ↓% is reported in integer in original paper.

Model	Method	Base Top-1	Pruned Top-1	Top-1 Acc. ↑(%)	FLOPs ↓%
VGG-16	FPGM	93.58	93.54	-0.04	34.20
	NS*	93.88	93.62	0.26	51.00
	NSP*	93.88	93.92	0.04	54.00
	AutoPruner	92.11	90.75	-1.36	52.98
	Ours(r=0.3)	92.11	94.17	**0.28**	**56.65**
ResNet-56	CCP*	93.50	93.69	0.19	47.00
	AutoPruner	93.89	93.83	-0.06	44.15
	Ours(r=0.4)	93.89	94.08	**0.19**	**47.68**
	AMC*	92.80	91.90	-0.90	50.00
	CP*	92.80	91.80	-1.00	50.00
	FPGM	93.59	93.26	-0.33	52.60
	SFP	93.59	93.35	-1.33	52.60
	ResRep	93.71	93.71	0.00	52.91
	AutoPruner	93.89	93.16	-0.73	52.83
	Ours(r=0.3)	93.89	94.32	**0.43**	**54.01**

Table 2
Performance comparisons on CIFAR-100. Top-1 Acc. ↑ is the Top-1 Accuracy gap between the pruned model and the baseline model. * means for this method, the FLOPs ↓% is reported in integer in original paper.

Model	Method	Base Top-1	Pruned Top-1	Top-1 Acc. ↑(%)	FLOPs ↓%
VGG-16	NS*	73.83	74.20	0.37	38.00
	COP*	72.59	71.77	-0.82	43.00
	NSP*	73.83	74.25	**0.42**	43.00
	AutoPruner	72.44	69.55	-2.89	43.66
	Ours(r=0.4)	72.44	72.78	0.34	**45.00**
ResNet-56	NS*	72.49	71.40	-1.09	24.00
	NSP*	72.49	72.46	-0.06	**25.00**
	AutoPruner	79.41	78.53	-0.88	23.32
	Ours(r=0.6)	79.41	79.50	**0.09**	24.61

an adequate speed-up ratio of 24.61%. In general, our method is among the best on CIFAR-100 compared to state-of-the-art methods. It seems that with medium size dataset, as the Flops drop increases, SP loses the winning margin in Top-1 Accuracy. According to the training log, we find the global sort starting to crumble on CIFAR-100 with a smaller compression rate, the performance is severely undermined. Hence, we believe current performance may be boosted using layer sort instead of global sort.

4.5. Comparisions on ILSVRC-12

For ILSVRC-12, our method outperforms state-of-the-art methods with $r = 0.4$ on ResNet-56 as shown in Table 3. Our method achieves the most FLOPs drop, 59.23, among all methods. Our Top-1 accuracy is slightly inferior to the current best, ResRep, but better than the rest. The common design shared by ResRep and us is that we apply an additional layer after the layer of interest and decide on neuron importance based on the gradient of the additional layer. ResRep uses Lasso loss to find the indicator mask in the "Res" part and uses the "Rep" part to ensure that the layer of interest is not over-pruned. Compared to them, our algorithm lacks the "Rep" part and thus, the network may be over-pruned at a given compression ratio. This postulation also corroborates that our algorithm usually achieves the best FLOPs drop but less stunning Top-1 Accuracy.

On MobileNetV2, SP achieves the most FLOPs drop, but the Top-1 accuracy decreases significantly compared to other algorithms. MobileNetV2 has repeating blocks with different hidden layer output channels. Hence, we choose to prune block after block for all seven blocks to maximize model performance. Based on our

Table 3
Performance comparisons on ILSVRC-12. Top-1 Acc. ↑ is the Top-1 Accuracy gap between the pruned model and the baseline model.

Model	Method	Base Top-1	Pruned Top-1	Top-1 Acc. ↑(%)	FLOPs ↓%
ResNet-56	SFP	76.15	74.61	-1.54	41.80
	MetaPruning	76.60	75.40	-1.20	51.10
	ResRep	76.15	76.15	**0.00**	54.54
	Adapt-DCP	76.01	75.15	-0.86	52.41
	C-SGD (extension)	76.15	75.29	-0.86	55.44
	ABCPruner	76.01	73.52	-2.49	56.61
	AutoPruner	76.15	74.76	-1.39	51.21
	Ours(r=0.4)	76.13	75.41	-0.72	**59.23**
MobileNetV2	MetaPruning	72.00	71.20	-0.80	27.67
	AMC	71.80	70.80	-1.00	29.67
	DMCP	72.30	72.20	**-0.10**	29.67
	AutoPruner	72.19	71.18	-1.01	30.87
	Ours(r=0.4)	71.88	70.40	-1.48	**33.41**

Table 4
Performance comparisons between SNIP and SP on CIFAR-10, CIFAR-100 and ILSVRC-12. Top-1 Acc. ↑ is the Top-1 Accuracy gap between the pruned model and the baseline model. RS, V, MB are abbreviations for ResNet-56, VGG-16, and MobileNetV2 structure. * indicates the model is finetuned, otherwise single-shot.

Datasets	Methods	Base Top-1	Pruned Top-1	Top-1 Acc. ↑(%)	Params Drop (%)
CIFAR-10	SNIP:RS	93.89	33.78	-60.11	-
	SNIP:RS*	93.89	94.00	0.11	59.87
	Ours:RS	93.89	94.08	**0.19**	59.03
	SNIP:V	92.11	10.00	-82.11	-
	SNIP:V*	92.11	92.16	0.05	79.90
	Ours:V	92.11	92.39	**0.28**	87.69
CIFAR-100	SNIP:RS	79.41	3.24	-76.17	-
	SNIP:RS*	79.41	79.06	-0.35	39.60
	Ours:RS	79.41	79.50	**0.09**	47.70
	SNIP:V	72.44	4.90	-67.54	-
	SNIP:V*	72.44	72.55	0.11	69.11
	Ours:V	72.44	72.78	**0.34**	68.51
ILSVRC-12	SNIP:RS	76.15	0.31	-75.84	-
	SNIP:RS*	76.15	74.36	-1.79	48.98
	Ours:RS	76.13	75.41	**-0.72**	50.30
	SNIP:MB	71.88	0.11	-71.77	-
	SNIP:MB*	71.88	68.93	-2.95	23.21
	Ours:MB	71.88	70.40	**-1.48**	23.82

training log, we discover that the model performance keeps dwindling until the 5th block with a Top-1 Accuracy of 55.62%. As we continue to prune blocks 6 and 7, the Top-1 Accuracy rebounds to 61.07% and 62.87% respectively. Despite this final surge, we notice that the Top-1 Accuracy after the 1st epoch in fine-tuning stage drops to 57.90%, which is significantly lower than before fine-tuning. Hence, we believe that the less optimal results for MobileNet2 are not due to our pruning algorithm, but because our fine-tuning stage requires further hyper-parameter tuning.

4.6. Comparisons with SNIP

SNIP is an unstructured pruning algorithm, which requires a mini-batch to generate the strategy and fine-tuning to restore the model performance. In the current hardware setting, only "pseudo" pruning is enabled, and thus, the FLOPs Drop for SNIP is not available. Therefore, we display the performance in a separate table with the Parameter Drop Rate as the substitute for the FLOPs Drop rate. In the original paper, the SNIP is applied to a model with randomized weight initiation. For a fair comparison, we use pre-trained models in SNIP strategy derivation.

From Table 4, we observe that the single-shot strategy, as an unstructured pruning method, is indeed very coarse in determining the best pruning strategy, considering that we use a pre-trained model in the derivation process. After fine-tuning, the performance of SNIP significantly improves but remains inferior to our method. In smaller datasets, it seems that SNIP is still comparable to ours, but as the datasets get more complex, the SNIP strategy becomes less competent. We notice that in smaller datasets, a mini-batch of data is more representative than the same batch in large datasets because a higher proportion of data is utilized to generate the strategy. Using SNIP importance measure with a mini-batch in a single shot suffers from sampling bias intensely, and the SP has reduced sampling bias with the "training-while-finetuning" framework.

5. Ablation studies

5.1. Performance with different compression rates

To study the effect of different compression rates r, we prune 20%, 30%, 40%, 50%, 60%, 70%, 80% channels from ResNet-56 and VGG-16 and evaluate on CIFAR-10. Notice that compression rates r indicate the percentage of channels kept to be active. The experimental results are shown in Fig. 4. In general, the lower the r gets, the more channels are pruned, and hence, worse performance should be expected. On smaller datasets, such as CIFAR-10, we notice that pruning up to 70% channels will result in better performance than the pre-trained model, and the best accuracy is achieved at $r = 0.4$, which means 60% of channels are removed. This phenomenon suggests that both VGG-16 and ResNet-56 are overfitted on CIFAR-10, and thus, the removal of channels can mitigate the overfitting effect and achieve better results. We notice that the FLOPs drop has a sudden peak at $r = 0.6$ on VGG-16, which means that the network selects kernels at deeper layers to be pruned to achieve the intended model size. We reckon that this peak can be attributed to the decrease in r from 0.7 to 0.6 pushing the strategy generation network to assign higher importance to shallow kernels to ensure accuracy. Therefore, more deep kernels have been pruned which will not contribute to the FLOPs drop much.

We also pruned 50%, 60%, 70%, 80%, 90% channels from ResNet-56 on ILSVRC-12. The results are displayed in Fig. 5. As the compression rate decreases, more channels are pruned, and hence performance declines and FLOPs drop increases. At $r = 0.5$, the Top-1 accuracy for Layer Sort exceeds the pre-trained model accuracy by 1.07%, but the FLOPs drop plummets that only 7.49% drop is reached. A similar pattern is also found in global sort models, which means that the ResNet-56 is less overfitted on ILSVRC-12, and pruning up to about half the channels should eliminate the overfitting effect completely. It is interesting to find out that at $r = 0.5$, the global sort can outdo the layer sort in terms of the FLOPs drop. Based on 6, we discover that the layer sort prune more channels in group 1 and 2, which contain all the shallower blocks, but it fails to prune group 3 and 4. Particularly, in group 4, not a single channel is pruned by layer sort, but it was heavily pruned by global sort, and hence, this discrepancy is manifested in the FLOPs drop.

On ILSVRC-12, ResNet-56 seems to be the adequate model, because as more channels were pruned, smaller models offered waning performances. Interestingly, $r = 0.4$ seems to be the best compression rate for ResNet-56 that not only does it achieve the best performance on CIFAR-10, but it also produces the best results across current models available with the FLOPs drop intended. It

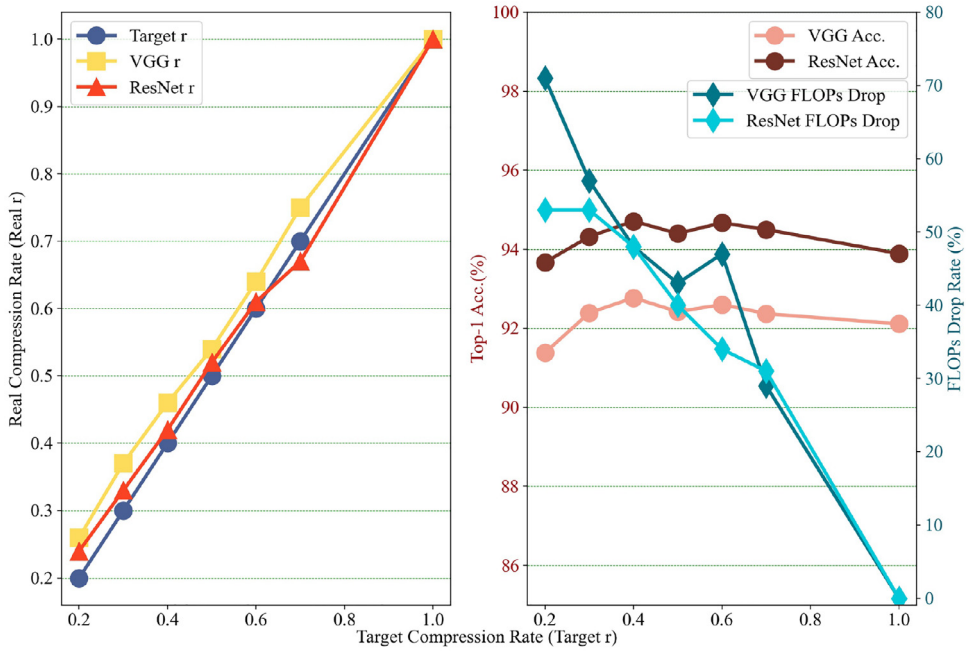

Fig. 4. ResNet-56 and VGG-16 on CIFAR-10. On the left is the comparison between the target and the real compression rate; on the right is the Top-1 Acc & FLOPs drop with different compression rates *r*. Accuracy is reported w.r.t the left y-axis; FLOPs drop is reported w.r.t the right y-axis.

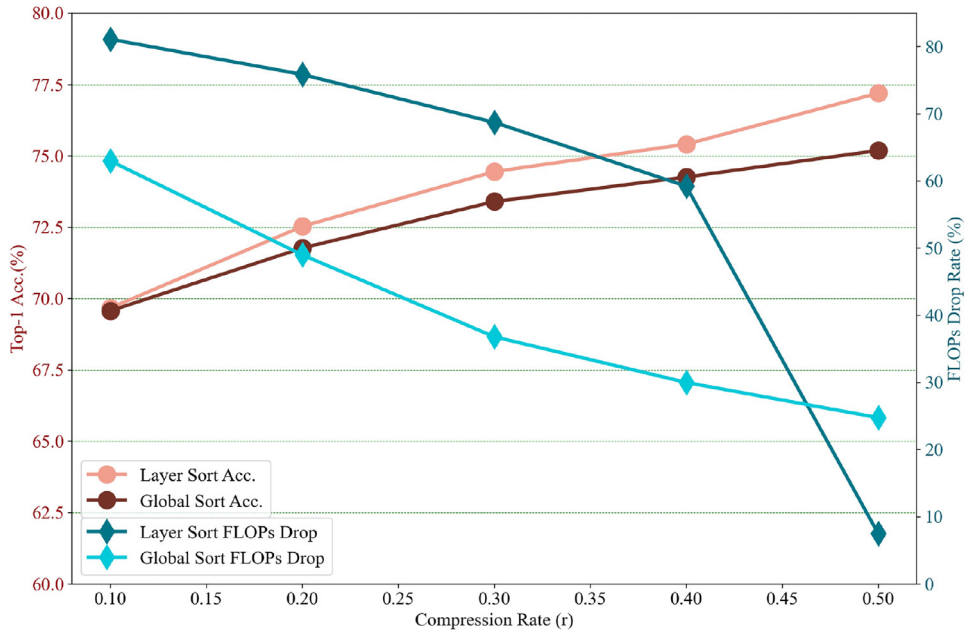

Fig. 5. Global vs. Layer sorting using ResNet-56. Accuracy is reported w.r.t the left y-axis; FLOPs drop is reported w.r.t the right y-axis.

seems that the optimal compression rate for ResNet-56 for our method is 0.4.

5.2. Effect of different β initiations

β initiation is crucial for the convolutional kernel to learn the pruning strategy. With β being too big or too small, we may end up with a random selection kernel or prolonged convergence. Different network structures require distinct β initiation, and we demonstrate the effect of initiation using VGG-16 on CIFAR-10.

The results in Table 5 demonstrate that different initiation has little effect on FlOPs or Params drop, but has some effect on the Top-1 accuracy. Using $\beta = 0.05$, the VGG-16 achieved an improved Top-1 accuracy by 0.28, compared to the second-best performance using $\beta = 0.1$. Using MobileNetV2 on ILSVRC-12, we discover that

Table 5
Effect of β initiation for VGG-16 on CIFAR-10.

β	FLOPs ↓%	Params ↓%	Top-1 Acc.(%)
0.005	47%	80.11%	92.00
0.01	47%	80.13%	92.46
0.05	48%	80.43%	**93.05**
0.1	48%	80.26%	92.77
0.2	47%	80.21%	91.06

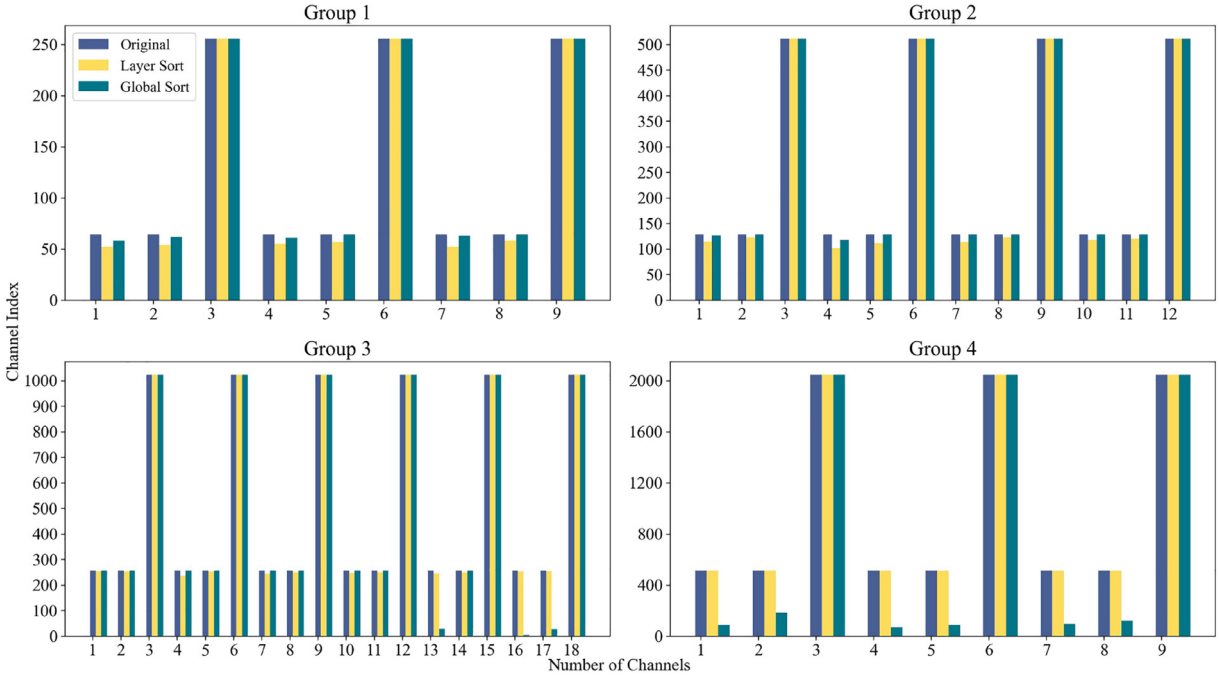

Fig. 6. Global vs. Layer sorting using ResNet-56. Comparing the number of channels pruned in each layer per group.

Table 6
Global vs. Layer sorting, MobileNetV2 and VGG-16 on ILSVRC-12.

Model	Sorting	Compre-ssion Rate	Base Top-1	Pruned Top-1	Top-1 Acc. ↑(%)	FLOPs ↓%
MobileNetV2	Global	0.20	71.88	58.77	-13.11	45.54
	Layer	0.20	71.88	64.76	-7.11	63.59
VGG-16	Global	0.20	73.36	71.56	-1.80	22.21
	Global	0.40	73.36	74.29	0.93	12.00
	Global	0.60	73.36	75.02	1.66	10.49

β initiation is also worth tuning. The performance of the pruned models improved by 0.11% with β initiations changing from 1 to 0.05.

5.3. Effect of global sort versus layer sort

To decide which output channel should be pruned, we use global sort to find a threshold, and the channel with a sensitivity smaller than the threshold will be pruned. This strategy takes all layers to be pruned into consideration and tries to save the most important channels across layers. Inevitably, it also leads to a discrepancy between the target compression rate and the real compression rate, as shown in Fig. 4. Despite the discrepancy, global sort works well on CIFAR-10 and CIFAR-100 for VGG-16 and ResNet-56. On ILSVRC-12, global sort favors channels in shallow layers and prunes channels in deeper layers heavily. With a large compression rate r, the pruning strategy attempts to reach the goal by pruning almost all the channels in deeper layers and producing much worse accuracy and minor FLOPs drop, because shallow channels are all kept. Thus, we introduce layer sort, which calculates the threshold for each layer and prunes channels to meet the preset r individually.

As shown in Fig. 5, ResNet-56 achieves better results on ILSVRC-12 using the layer sort strategy. Based on our experiments, MobilNetV2 also obtains a better pruning strategy using the layer sort strategy. The Top-1 accuracy improves and the FLOPs drop more. These effects can be attributed to the fact that for ILSVRC-12, both ResNet-56 and MobileNetV2 are adequate models, which do not have much overfitting effect on CIFAR-10. Moreover, it seems that global sort still achieves good Top-1 accuracy, though not ideal FLOPs drop, using VGG-16 on ILSVRC-12, as shown in Table 6. The discrepancy in accuracy among models can be attributed to that for network structures scrupulously designed to have repeated block structures, such as ResNet-56 and MobileNetV2, global sort fails to balance between shallow and deep kernel importance, but for less complicated layer-stacked structures, such as VGG-16, such effect is less significant.

5.4. Sampling bias of single-shot strategy

The single-shot strategy proposed by SNIP is used in our method to guide the learning of the pruning strategy. We presume that the single-shot strategy points in the right direction but is biased by the batch used. In this section, we would like to demonstrate that the single-shot strategy is susceptible to sampling bias and that the learned strategy can converge to it without being misguided.

In Fig. 7, we report the agreement rate between the single-shot strategy and learned strategy every 20 steps for 4 epochs of training. The agreement rate is defined with the single-shot mask $\mathbf{M}_{ss}$, the learned mask $\mathbf{M}_{le}$, and the number of channels C as:

$$Agreement = \frac{\mathbf{M}_{ss} \cap \mathbf{M}_{le}}{C}. \tag{13}$$

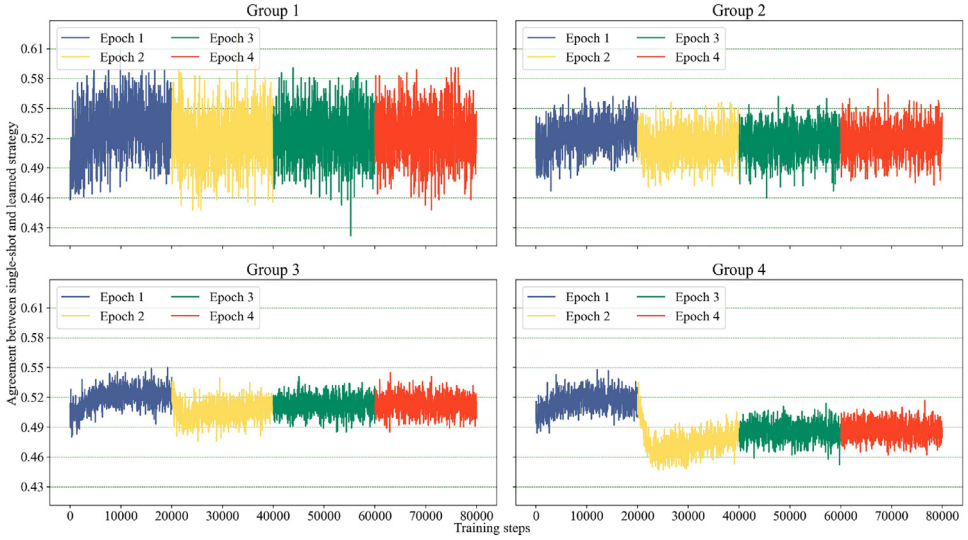

Fig. 7. Agreement rate between single-shot strategy and learned strategy.

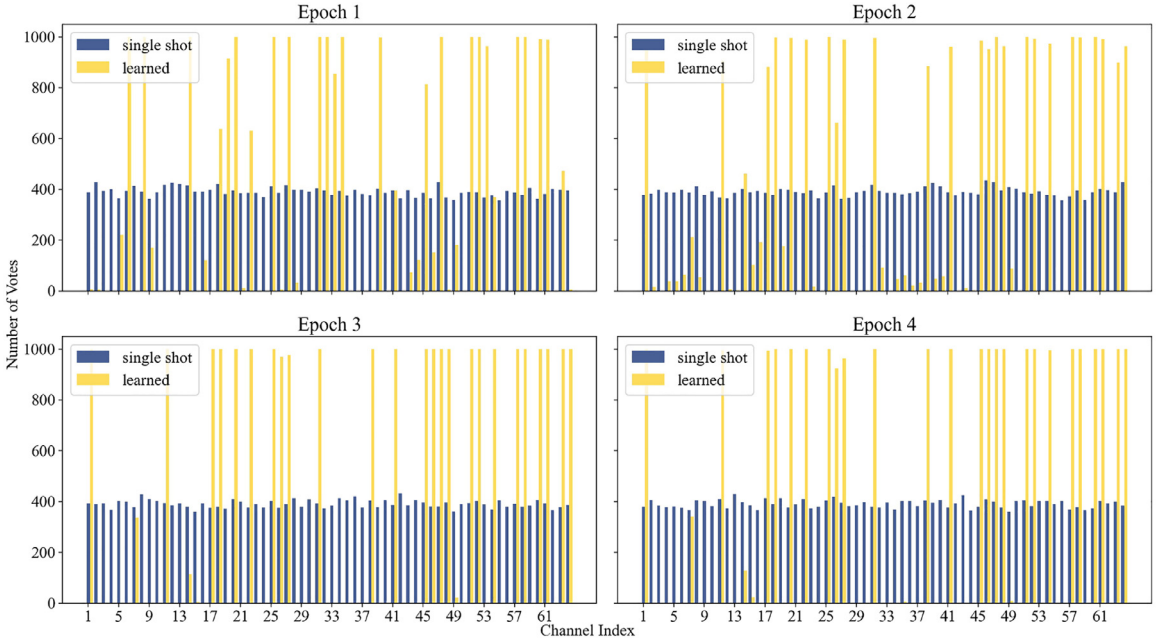

Fig. 8. Batch Votes on single-shot and learned strategy on layer 1 in group 1.

Using ResNet-56 with $r = 0.4$ on ILSVRC-12, we group layers of interest into four groups and each group contains 9, 12, 18, and 9 layers respectively. The agreement rate is averaged over group. For group 1 and 2, the agreement rate does not seem to change as epochs increase. For group 3 and 4, the agreement rate drops as training steps grow. The learned strategy, which is the final strategy applied, does not agree with the single-shot strategy completely, yet does not deviate from it much. Therefore, the guidance provided by the single-shot strategy is crucial, and the learned strategy can remove sampling bias as we train for more steps.

A channel to be pruned or active is denoted by 0 and 1 respectively. Hence, every 20 steps, we accumulate the single-shot and learned mask to indicate which channel index is voted to be pruned or active the most. Fig. 8 is intended to demonstrate the batch votes for layer 1 in group 1. From the images, we discover that the learned votes favor certain channels strongly and the single-shot votes reveal no preference.

We also calculate the agreement rate between two strategies w.r.t. pruning mask and active mask respectively. Fig. 9 suggests that the agreement between group 1 and 2 on whether the channel should be pruned or active increases over epochs. On the contrary, the agreement in group 3 and 4 decreases as the training step increases.

The channels in the single-shot strategy, supported by most batches, are also more likely to remain in the final strategy. It can be attributed that the learned strategy can be taught by the single-shot strategy and can also prevent sampling bias.

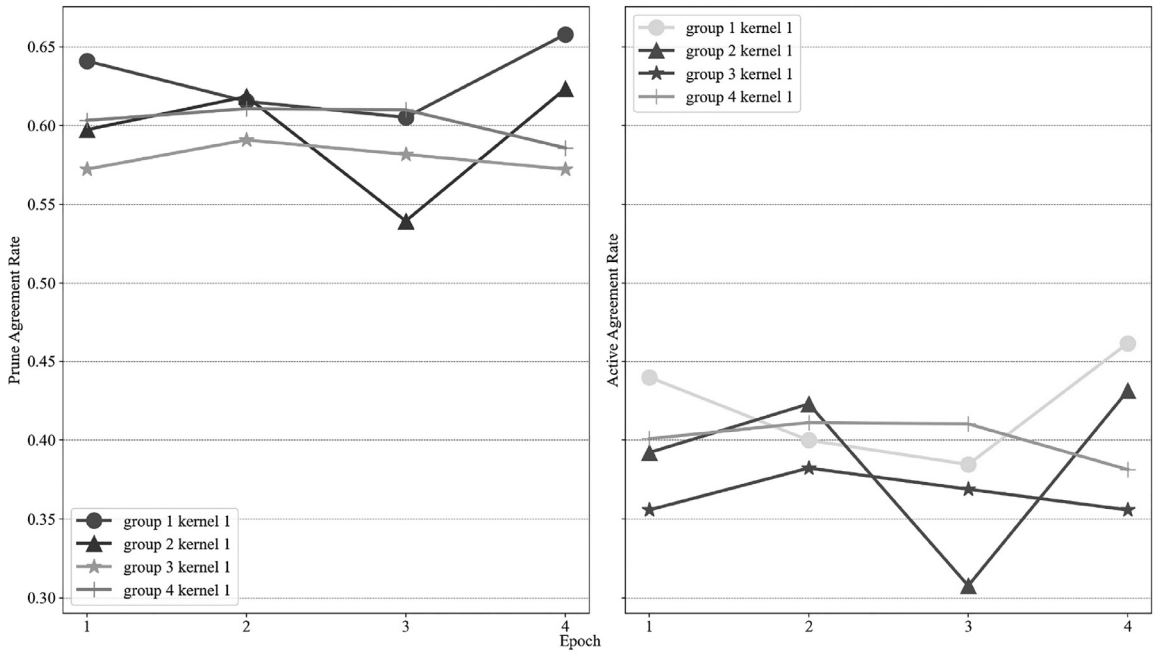

Fig. 9. Strategy agreement on prune or active. The agreement is calculated with $r = 0.4$ using the first kernel in each group.

6. Conclusion

In this paper, we have proposed a Sensitivity Pruner(SP) using the "pruning while fine-tuning" framework to compress deep neural networks. We first recognize that SNIP offers a great metric for neuron importance but is susceptible to sampling bias if used in a single-shot as suggested. Then, we integrate this metric into the "pruning while fine-tuning" framework to eliminate the sampling bias and improve the framework with better sensitivity measurement. To select channels to be pruned, we use the single-shot strategy to guide the learning of the pruning strategy and dynamically balance the training goal between model accuracy and intended compression rate with a self-adaptive hyper-parameter λ. Extensive results on image classification have verified the effectiveness of the proposed method.

Compared to the single-shot strategy, not only does our algorithm allow all-batch assessment, but also enables the pruning strategy to be delivered block-by-block, so that after the compression has been implemented in previous blocks, the current block can be pruned according to the latest sparsity. Besides overcoming the sampling bias in the single-shot strategy, we also make the pruning strategy to be derived dynamically, which improves the pruned network performance with the provided saliency measure. Moreover, the unstructured pruning from SNIP is adapted by us to enable channel-wise structured pruning. Without additional hardware support, the compressed model can still enjoy the speed-up, not just the "pseudo" pruning as in SNIP.

Compared to the Autopruner framework, we have innovated the neuron importance measure and re-defined the loss function so that the pruning strategy not only focuses on retaining accuracy but also strives to prune as many channels as intended with the guidance of the single-shot strategy. Thus, our method can utilize the SNIP saliency measure as the target and the convolutional kernel $\boldsymbol{U}$ to generalize strategy to all batches, which equips the framework with a more substantial importance measure than extracting pruning strategy from feature maps. The final algorithm can be implemented in any network structure consisting of convolutional or linear layers and derive the pruning strategy automatically with an arbitrary pruning ratio.

Besides strengths, the limitations of our strategy should also be mentioned. First, the β initiation is a hyper-parameter to be tuned. Second, our experiments generally focus on computer vision classification tasks, which can be less informative considering other tasks, such as time series forecasting. Third, although our algorithm holds no assumption for datasets, we observe that with bigger datasets, the performance is not as outstanding as with small datasets.

In the future, we may improve this method from two perspectives. First, the initiation of the β should be automated so that any network structure can be easily pruned with fewer trials for better performance. Second, we will extend the proposed SP for model quantization. Particularly, we can apply the quantization for each selected channel before pruning the next channel, which allows us to perform channel pruning and quantization simultaneously.

Moreover, we would like to point out that our algorithm can be extended to other popular networks, such as the attention mechanism in the transformer. In the attention mechanism, the linear projections for key, query, and value can be pruned to prevent redundancy in the attention map, which has been proved to be a problem by previous studies. It is worth noting that the attention matrix calculated by the multiplication of key and query, outputted from the pruned linear layers, may contain too many masked values, rendering it too sparse. We recommend future research to dive into this issue and modify our proposed method to the attention mechanism. Moreover, more experiments can be done on other big datasets to verify our observation and possibly find the rationale behind it and improve the algorithm to better accommodate bigger datasets. Lastly, the theoretical proof of the effectiveness of pruning methods is generally lacking, which is also a promising research direction.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the link to my code at the Attach Files step.

Acknowledgments

This work was supported in part by the STI 2030-Major Projects of China under Grant 2021ZD0201300, and by the National Science Foundation of China under Grant 62276127.

References

[1] I. Manipur, M. Manzo, I. Granata, M. Giordano, L. Maddalena, M.R. Guarracino, Netpro2vec: a graph embedding framework for biomedical applications, IEEE/ACM TCBB 19 (2) (2022) 729–740.
[2] Y. Cheng, D. Wang, P. Zhou, T. Zhang, Model compression and acceleration for deep neural networks: the principles, progress, and challenges, IEEE Signal Process. Mag. 35 (1) (2018) 126–136.
[3] H. Li, H. Samet, A. Kadav, I. Durdanovic, H.P. Graf, Pruning Filters for efficient ConvNets, ICLR, 2017.
[4] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, W. Lin, Thinet: pruning CNN filters for a thinner net, TPAMI 41 (10) (2019) 2525–2538.
[5] J. Liu, B. Zhuang, Z. Zhuang, Y. Guo, J. Huang, J. Zhu, M. Tan, Discrimination-aware network pruning for deep model compression, TPAMI 44 (8) (2022) 4035–4051.
[6] Y. He, P. Liu, Z. Wang, Z. Hu, Y. Yang, Filter pruning via geometric median for deep convolutional neural networks acceleration, in: CVPR, 2019, pp. 4340–4349.
[7] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, S. Han, AMC: AutoML for model compression and acceleration on mobile devices, in: ECCV, 2018, pp. 815–832.
[8] J.-H. Luo, J. Wu, Autopruner: an end-to-end trainable filter pruning method for efficient deep model inference, Pattern Recognit. 107 (2020) 107461.
[9] N. Lee, T. Ajanthan, P.H. Torr, SNIP: single-shot network pruning based on connection sensitivity, ICLR, 2019.
[10] T. Zhuang, Z. Zhang, Y. Huang, X. Zeng, K. Shuang, X. Li, Neuron-level structured pruning using polarization regularizer, in: NeurIPS, volume 33, 2020, pp. 9865–9877.
[11] C. Tai, T. Xiao, X. Wang, W. E, Convolutional neural networks with low-rank regularization, in: Int. Conf. Learn. Represent., 2016.
[12] V. Lebedev, Y. Ganin, M. Rakhuba, I.V. Oseledets, V.S. Lempitsky, Speeding-up Convolutional neural networks using fine-tuned CP-decomposition, ICLR, 2015.
[13] T. Cohen, M. Welling, Group Equivariant convolutional networks, in: ICML, 2016, pp. 2990–2999.
[14] W. Shang, K. Sohn, D. Almeida, H. Lee, Understanding and improving convolutional neural networks via concatenated rectified linear units, in: ICML, 2016, pp. 2217–2225.
[15] A. Romero, N. Ballas, S.E. Kahou, A. Chassang, C. Gatta, Y. Bengio, FitNets: hints for thin deep nets, ICLR, 2015.
[16] P. Luo, Z. Zhu, Z. Liu, X. Wang, X. Tang, Face model compression by distilling knowledge from neurons, in: AAAI, 2016, pp. 3560–3566.
[17] T. Chen, I.J. Goodfellow, J. Shlens, Net2Net: accelerating learning via knowledge transfer, ICLR, 2016.
[18] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision, in: ICML, 2015, pp. 1737–1746.
[19] M. Courbariaux, Y. Bengio, J.-P. David, BinaryConnect: training deep neural networks with binary weights during propagations, in: NeurIPS, 2015, pp. 3123–3131.
[20] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, XNOR-Net: ImageNet classification using binary convolutional neural networks, in: ECCV, 2016, pp. 525–542.
[21] J.M. Alvarez, M. Salzmann, Learning the number of neurons in deep networks, in: NeurIPS, 2016, pp. 2262–2270.
[22] X. Zhu, W. Zhou, H. Li, Improving Deep neural network sparsity through decorrelation regularization, in: IJCAI, 2018, pp. 3264–3270.
[23] S. Han, H. Mao, W.J. Dally, Deep Compression: Compressing deep neural network with pruning, trained quantization and huffman coding, in: Y. Bengio, Y. LeCun (Eds.), ICLR, 2016.
[24] S. Han, J. Pool, J. Tran, W.J. Dally, Learning both weights and connections for efficient neural network, in: NeurIPS, 2015, pp. 1135–1143.
[25] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: ICCV, 2017, pp. 1398–1406.
[26] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, ICLR, 2015.
[27] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: CVPR, 2016, pp. 770–778.
[28] M. Sandler, A.G. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, MobileNetV2: inverted residuals and linear bottlenecks, in: CVPR, 2018, pp. 4510–4520.
[29] A. Krizhevsky, Learning multiple layers of features from tiny images, Technical Report, 2009.
[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: a large-scale hierarchical image database, in: CVPR, 2009, pp. 248–255.
[31] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning Efficient convolutional networks through network slimming, in: ICCV, 2017, pp. 2755–2763.
[32] H. Peng, J. Wu, S. Chen, J. Huang, Collaborative channel pruning for deep networks, in: ICML, 2019, pp. 5113–5122.
[33] Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, Soft filter pruning for accelerating deep convolutional neural networks, in: IJCAI, 2018, pp. 2234–2240.
[34] X. Ding, T. Hao, J. Tan, J. Liu, J. Han, Y. Guo, G. Ding, ResRep: lossless CNN pruning via decoupling remembering and forgetting, in: ICCV, 2021, pp. 4490–4500.
[35] W. Wang, C. Fu, J. Guo, D. Cai, X. He, COP: customized deep model compression via regularized correlation-based filter-level pruning, in: IJCAI, 2019, pp. 3785–3791.
[36] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, J. Sun, MetaPruning: meta learning for automatic neural network channel pruning, in: ICCV, 2019, pp. 3295–3304.
[37] X. Ding, T. Hao, J. Han, Y. Guo, G. Ding, Manipulating identical filter redundancy for efficient pruning on deep and complicated CNN, 2021. arXiv:2107.14444
[38] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, Y. Tian, Channel pruning via automatic structure search, in: IJCAI, volume 1, 2020, pp. 673–679.
[39] S. Guo, Y. Wang, Q. Li, J. Yan, DMCP: differentiable markov channel pruning for neural networks, in: CVPR, 2020, pp. 1536–1544.

Suhan Guo received the B.A degree from The Johns Hopkins University, Maryland, U.S., in 2016 and MPH degree from New York University, New York, U.S., in 2019. She is currently pursuing the Ph.D. degree at Nanjing University, Nanjing, China. Her current research interests include times series forecasting and neural network pruning.

Bilan Lai received the B.Sc. degree from South China Agricultural University of Computer Science and Technology, Nanjing, China, in 2018, and received the M.Sc. degree from the Nanjing University of Computer Science and Technology, Nanjing, China, in 2022. She currently works on computer vision related to 3D reconstruction and neural network pruning.

Suorong Yang received the B.Sc. degree from Nanjing University, Nanjing, China, in 2019, and is currently pursuing the Ph.D. degree at the Nanjing University, Nanjing, China. His current research interests include neural network pruning and computer vision.

Jian Zhao (Senior Member, IEEE) received the B.S. degree from Nanjing University, Nanjing, China, the M.Sc. degree from the Hamburg University of Technology, Hamburg, Germany, and the Dr. Sc. degree in electrical engineering from the Swiss Federal Institute of Technology (ETH) Zurich, Switzerland. From 2010 to 2015, he was a Research Scientist with the Institute for Infocomm Research, A*STAR, Singapore. Currently, he is an Associate Professor with the School of Electronic Science and Engineering, Nanjing University. His research interests include deep neural networks, mathematical optimization, and wireless communication networks. Dr. Zhao was honored with the Dengfeng Scholars Program of Nanjing University in 2015, IEEE Globecom 2008 Best Paper Award, and the 2009 Chinese Government Award for Outstanding Self-Financed Students Abroad.

Furao Shen (Member, IEEE) received the B.Sc. and M.Sc. degrees in mathematics from Nanjing University, Nanjing, China, in 1995 and 1998, respectively, and the Ph.D. degree from the Tokyo Institute of Technology, Tokyo, Japan, in 2006. He is currently a Full Professor of computer science and technology with Nanjing University. His current research interests include neural computing and robotic intelligence.