DPFPS: Dynamic and Progressive Filter Pruning for Compressing Convolutional Neural Networks from Scratch

Xiaofeng Ruan[1,2*], **Yufan Liu**[1,2*], **Bing Li**[1,4†], **Chunfeng Yuan**[1], **Weiming Hu**[1,2,3]
[1] National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences
[2] School of Artificial Intelligence, University of Chinese Academy of Sciences
[3] CAS Center for Excellence in Brain Science and Intelligence Technology
[4] PeopleAI Inc.
{ruanxiaofeng2017, yufan.liu}@ia.ac.cn, {bli, cfyuan, wmhu}@nlpr.ia.ac.cn

Abstract

Filter pruning is a commonly used method for compressing Convolutional Neural Networks (ConvNets), due to its friendly hardware supporting and flexibility. However, existing methods mostly need a cumbersome procedure, which brings many extra hyper-parameters and training epochs. This is because only using sparsity and pruning stages cannot obtain a satisfying performance. Besides, many works do not consider the difference of pruning ratio across different layers. To overcome these limitations, we propose a novel dynamic and progressive filter pruning (DPFPS) scheme that directly learns a structured sparsity network from Scratch. In particular, DPFPS imposes a new structured sparsity-inducing regularization specifically upon the expected pruning parameters in a **dynamic** sparsity manner. The dynamic sparsity scheme determines sparsity allocation ratios of different layers and a Taylor series based channel sensitivity criteria is presented to identify the expected pruning parameters. Moreover, we increase the structured sparsity-inducing penalty in a **progressive** manner. This helps the model to be sparse gradually instead of forcing the model to be sparse at the beginning. Our method solves the pruning ratio based optimization problem by an iterative soft-thresholding algorithm (ISTA) with dynamic sparsity. At the end of the training, we only need to remove the redundant parameters without other stages, such as fine-tuning. Extensive experimental results show that the proposed method is competitive with 11 state-of-the-art methods on both small-scale and large-scale datasets (i.e., CIFAR and ImageNet). Specifically, on ImageNet, we achieve a 44.97% pruning ratio of FLOPs by compressing ResNet-101, even with an increase of 0.12% Top-5 accuracy. Our pruned models and codes are released at https://github.com/taoxvzi/DPFPS.

Introduction

With the development of computer science and artificial intelligence, ConvNets have significantly improved performance in several fields, such as image classification (Krizhevsky, Sutskever, and Hinton 2012), object detection (Girshick et al. 2014), object tracking (Bertinetto et al. 2016) and video understanding (Venugopalan et al. 2015). However, a large number of parameters and FLoating-point OPerations (FLOPs) make it difficult to deploy ConvNets in the application of embedded or mobile devices. Therefore, model compression and acceleration of ConvNets is a fundamental problem that has been extensively studied in recent years. Several techniques, including weight sparsity, low-rank approximation, parameter quantization, and filter pruning, have been applied to the study of model compression and acceleration. Among them, filter pruning methods have been widely explored and possessed excellent performances on the compression and acceleration ratio because of their hardware-friendly capabilities without the support of special libraries. For example, by using SSL (Wen et al. 2016), convolutional layer computation of AlexNet is accelerated on average 5.1x and 3.1x speedups against CPU and GPU, respectively, with off-the-shelf libraries.

Though previous filter pruning algorithms have achieved promising results, there still exist several problems. Specifically, a common procedure of filter pruning (Liu et al.

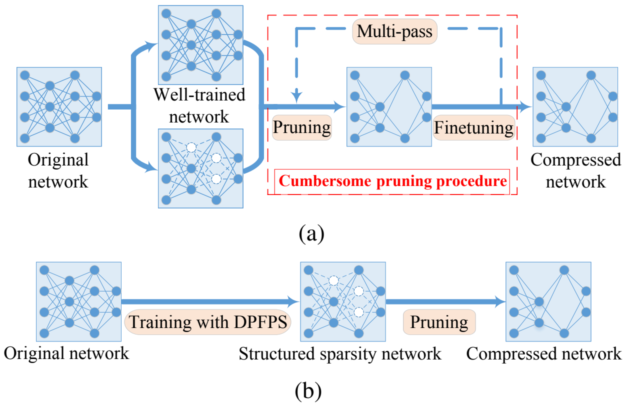


Figure 1: Block diagrams of two types of filter pruning. (a) This pruning procedure needs pre-trained models and fine-tuning after pruning, even a multi-pass scheme. (b) Our method directly learns a structured sparsity network from scratch, without any fine-tuning or multi-pass.

*Equal contribution.
†Corresponding author.

2019b) includes three stages: (1) training, (2) pruning, and (3) fine-tuning, as shown in Figure 1(a). Firstly, well-trained models usually need to be obtained. In some studies (Wen et al. 2016; Liu et al. 2017), structured sparsity networks are learned by imposing the structured sparsity regularization. Some other studies (Li et al. 2016; Luo, Wu, and Lin 2017) directly use available pre-trained models. Then, when well-trained models are pruned, there are two typical issues: (1) it is often difficult to match the pre-setting pruning ratio for the sparse rate of the network (after trained with a sparsity-inducing constrain), and (2) a dedicated handcrafted pruning criterion needs to be designed to identify the unimportant neurons and remove them. In order to achieve the pre-setting pruning ratio, some non-zero parameters are also removed. This is bound to bring accuracy drop. Hence, to restore performances, it is critical to fine-tune pruned models. However, when pruned models are fine-tuned, many hyper-parameters (e.g., the learning rate, epochs of fine-tuning, multi-pass times) need to be set again. This makes the multi-stage procedure to be more cumbersome.

To address the above issues, we propose a novel Dynamic and Progressive Filter Pruning (DPFPS) scheme that directly learns a structured sparsity network from Scratch, without pre-trained, fine-tuning, or multi-pass, as shown in Figure 1(b). In our proposed method, to meet the pre-setting pruning ratio, structured sparsity-inducing regularization is only imposed upon the expected pruning parameters, as shown in Figure 2. Different from fixed sparsity in each layer, our method dynamically updates sparsity allocation ratios in different layers. The dynamic sparsity scheme determines sparsity allocation ratios of different layers and a Taylor series based channel sensitivity criteria is presented to identify the expected pruning parameters. In this manner, reserved filters can have more space to learn sufficient information, leading to better performance. For redundant filters, the constraint can make them towards zero and boost the generation of a compact network. As a result, the dynamic sparsity allocation scheme makes it unnecessary to pre-define pruning architecture. Moreover, a group Lasso based progressive penalty is designed as the structured sparsity regularization, which increases gradually as training proceeds (i.e., from zero to a proper value). This encourages all the filters to learn useful information at the beginning and forces the redundant filters to be sparse in the late period. Our method solves the pruning ratio based optimization problem by an iterative soft-thresholding algorithm (ISTA) with dynamic sparsity. At the end of the training, we only need to remove the redundant parameters without other stages, such as fine-tuning, and the additional time consumption is also acceptable.

Our contributions are listed as follows:

- Our method **dynamically** updates sparsity allocation ratios in different layers and only imposes regularization upon the expected pruning parameters.
- Moreover, we design a group Lasso based **progressive** sparsity penalty to increasingly induce sparsity in expected pruning parameters.
- Our method solves the pruning ratio based optimization problem by an iterative soft-thresholding algorithm (ISTA) with dynamic sparsity. At the end of the training, we only need to remove the redundant parameters without other stages, such as fine-tuning. Extensive experimental results show that the proposed method is competitive with 11 state-of-the-art methods.

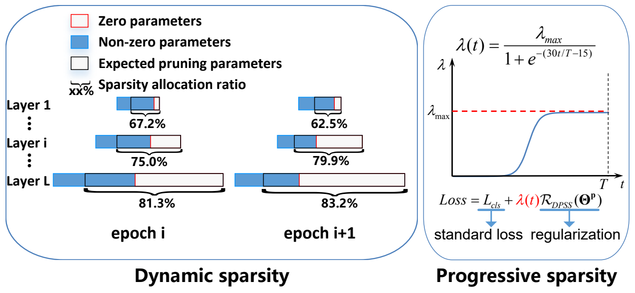


Figure 2: An illustration of DPFPS. (1) Sparsity allocation ratios and expected pruning parameters are dynamically updated. (2) The sparsity penalty increases gradually as training proceeds.

Related Work

Recent works on weight sparsity, parameter quantization, low-rank decomposition, filter pruning, and knowledge distillation can be found in some papers (Idelbayev and Carreira-Perpinan 2020; Jin, Yang, and Liao 2020; Liu et al. 2019a, 2020; Ruan et al. 2020). In this section, we review related works in filter pruning, which focus on two aspects of structured sparsity networks and fine-tuning.

Structured Sparsity Networks. Learning a structured sparsity network is a straightforward pruning method, which is widely used to compress and accelerate neural networks (Wen et al. 2016; Alvarez and Salzmann 2016; Singh et al. 2019; Lin et al. 2020b). Wen *et al.* (Wen et al. 2016) used group Lasso regularization to achieve a structured sparsity learning (SSL). Furthermore, a sparse group Lasso idea was applied to automatically determine the number of neurons in each layer of the network during learning (Alvarez and Salzmann 2016). Although a structured sparsity network was directly learned, the pruning ratio depended on the penalty term and the excessively pruned model needed to be fine-tuned to regain accuracy. In this paper, given a pre-setting pruning ratio, we directly learn a structured sparsity network from scratch and do not need any fine-tuning after pruning.

Fine-tuning. Due to hurting the performance after pruning the network, the original performance needed to be restored by fine-tuning techniques (Han et al. 2015). Some methods pruned the well-trained models by minimizing reconstruction error and then fine-tuned the pruned models (He, Zhang, and Sun 2017; Jiang et al. 2018). Although these methods have excellent convergence performance, the pruning process is often cumbersome, including obtaining the pre-trained model, pruning and recovering accuracy by minimizing layer-wise reconstruction error, and fine-tuning. Additionally, although the fine-tuning techniques (Ding et al. 2018; Yu et al. 2018; Molchanov et al. 2019; Peng et al. 2019; Ding et al. 2019a) are helpful to regain the original accuracy, many hyper-parameters need to be set and this is cumbersome.

The Proposed Method

In this section, we first introduce the problem formulation. Then, we provide details of progressive structured sparsity regularization. Finally, we present the dynamic sparsity algorithm.

Preliminary and Problem Formulation

In an L-layer ConvNet, the i-th convolutional layer parameters (ignoring biases) can be represented as a 4-dimensional tensor: $\mathcal{W}^{(i)} \in \mathbb{R}^{c^{(i)} \times r^{(i)} \times k_1^{(i)} \times k_2^{(i)}}$, where $c^{(i)}$ and $r^{(i)}$ are the numbers of output and input channels (feature maps), and $k_1^{(i)} \times k_2^{(i)}$ corresponds to the 2-dimensional spatial kernel. We reorganize the parameters from the original space $\mathcal{W}^{(i)} \in \mathbb{R}^{c^{(i)} \times r^{(i)} \times k_1^{(i)} \times k_2^{(i)}}$ to $(\mathbf{W}^{(i)})^{2D}_{out} \in \mathbb{R}^{c^{(i)} \times r^{(i)} k_1^{(i)} k_2^{(i)}}$ or $(\mathbf{W}^{(i)})^{2D}_{in} \in \mathbb{R}^{r^{(i)} \times c^{(i)} k_1^{(i)} k_2^{(i)}}$, where $(\mathbf{W}^{(i)})^{2D}_{out}$ and $(\mathbf{W}^{(i)})^{2D}_{in}$ denote the matrix-forms of tensor $\mathcal{W}^{(i)}$ along out and in channels, respectively. Then $\mathcal{W}^{(i)}_j$ can be rewritten as a 3-dimensional tensor: $\mathcal{K}^{(i)}_j \in \mathbb{R}^{r^{(i)} \times k_1^{(i)} \times k_2^{(i)}} (j = 1, ..., c^{(i)})$ to represent the j-th filter kernel in the i-th convolutional layer, which corresponds to the weights of the *j-th* output channel. Besides convolutional layers, the parameters of BatchNorm layers are denoted as $\{\gamma^{(i)}_j, \beta^{(i)}_j\} \in \mathbb{R}$, which scale and shift the normalized value. To integrate the above parameters, we define $\mathbf{\Theta}$ as all parameters set in the network, i.e. $\mathbf{\Theta} = \bigcup_{i \in N_1, j \in N_2} (\mathcal{K}^{(i)}_j \cup \{\gamma^{(i)}_j, \beta^{(i)}_j\})$, where $N_1 = \{1, ..., L\}, N_2 = \{1, ..., c^{(i)}\}$.

We consider pruning the network as a constrained optimization problem (Boyd and Vandenberghe 2004), given input-output pairs $(\mathbf{x}, \mathbf{y})$ from data set $\mathcal{D}$, which has the form

$$\begin{aligned} \min_{\mathbf{\Theta}} & \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \mathcal{L}(\mathcal{F}(\mathbf{\Theta}, \mathbf{x}), \mathbf{y}), \\ \text{s.t.} \quad & PR_0 - PR(\mathbf{\Theta}) \leq 0. \end{aligned} \tag{1}$$

where $\mathcal{F}(\cdot)$ is the ConvNet forward function, $\mathcal{L}(\cdot)$ is the standard loss function (e.g., the cross-entropy loss for classification tasks), $PR(\cdot)$ is an evaluation metrics, e.g., pruning ratio of parameters or FLOPs, and PR_0 is a pre-setting pruning ratio. It is not convenient to directly constrain $PR(\mathbf{\Theta})$, so we use the layer-wise form to rewrite the problem (1) as the unconstrained problem (Zhang et al. 2018)

$$\min_{\mathbf{\Theta}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \mathcal{L}(\mathcal{F}(\mathbf{\Theta}, \mathbf{x}), \mathbf{y}) + \sum_{i=1}^{L} t_i(\mathbf{W}^{(i)}), \tag{2}$$

where $t_i(\cdot)$ is an indicator function of $\mathbf{W}^{(i)}$, which has the form

$$t_i(\mathbf{W}^{(i)}) = \begin{cases} 0, & card(\mathbf{W}^{(i)}) \leq p^{(i)}, \\ +\infty, & otherwise. \end{cases} \tag{3}$$

where $card(\cdot)$ returns the number of nonzero of filters' ℓ_2-norm in its layer, i.e. the number of $\left\|\mathbf{W}^{(i)}_{j,:}\right\|_2 \neq 0$ (j row of $\mathbf{W}^{(i)}$), and $p^{(i)}$ denotes the desired number of preserved filters in the i-th layer. Because the second term of Equation (2) is not differentiable, the problem cannot be directly addressed by stochastic gradient descent.

Progressive Structured Sparsity Regularization

In previous works (Wen et al. 2016; Liu et al. 2017), structured sparsity regularization is widely used to introduce the objective function as an unconstrained optimization problem, which is formulated as:

$$\min_{\mathbf{\Theta}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \mathcal{L}(\mathcal{F}(\mathbf{\Theta}, \mathbf{x}), \mathbf{y}) + \lambda \mathcal{R}(\mathbf{\Theta}). \tag{4}$$

where $\mathcal{R}(\cdot)$ is the penalty function to make ConvNet be sparse and λ is the penalty coefficient. However, the hyper-parameter λ does not directly control the pruning ratio. To meet the pre-setting pruning ratio, the pruned model may need to be fine-tuned to recover the accuracy or obtained by some empirical knowledge (e.g., several attempts).

To address this weakness, different from previous sparsity strategy (Wen et al. 2016; Liu et al. 2017), during training, we control the pruning ratio like Equation (2) and impose the structured sparsity regularization upon the expected pruning filters like (4). Specifically, we divide parameters set $\mathbf{\Theta}$ into two subsets: $\mathbf{\Theta^P}$ and $\overline{\mathbf{\Theta^P}}$, where $\mathbf{\Theta^P}$ is the expected pruning filters that are "unimportant" and $\overline{\mathbf{\Theta^P}}$ is the preserved filters. Therefore, based on our proposed structured sparsity regularization, the loss function can be expressed as:

$$Loss = \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \mathcal{L}(\mathcal{F}(\mathbf{\Theta}, \mathbf{x}), \mathbf{y}) + \lambda \mathcal{R}_{DPSS}(\mathbf{\Theta^P}). \tag{5}$$

where $\mathcal{R}_{DPSS}(\cdot)$ is our designed dynamic and progressive structured sparsity regularization (DPSS), which only works on the expected pruning parameters $\mathbf{\Theta^P}$.

Inspired by the Group Lasso (Yuan and Lin 2006) method, we use ℓ_{21}-norm as the sparsity regularization, which has the form:

$$\begin{aligned} & \mathcal{R}_{DPSS}(\mathbf{\Theta^P}) = \\ & \sum_{i=1}^{L} \sum_{j=1}^{c^{(i)}-p^{(i)}} \left\|(\mathbf{W}^{(i)}_{j,:})^{2D}_{out}\right\|_2 + \sum_{i=1}^{L-1} \sum_{j=1}^{c^{(i)}-p^{(i)}} \left\|(\mathbf{W}^{(i+1)}_{j,:})^{2D}_{in}\right\|_2. \end{aligned} \tag{6}$$

When the channels are removed, parameters of both the current and the next layers are pruned. So, we impose the regularization upon the corresponding parameters in the current and the next layers.

In Equation (5), the second term $\mathcal{R}_{DPSS}(\mathbf{\Theta^P})$ is convex but not differentiable. In order to solve the non-smooth unconstrained optimization problem, we make use of ISTA (Beck and Teboulle 2009) and update $(\mathbf{W}^{(i)}_{j,m})^{2D} \in$

$\mathbf{\Theta^P}$ by $\{(\mathbf{W}^{(i)}_{j,\,m})^{2D}\}^{(n+1)} = S_\lambda(\{(\mathbf{W}^{(i)}_{j,\,m})^{2D}\}^{(n)})$, where

$$S_\lambda((\mathbf{W}^{(i)}_{j,\,m})^{2D}) = \begin{cases} (\mathbf{W}^{(i)}_{j,\,m})^{2D} - \frac{\lambda(\mathbf{W}^{(i)}_{j,\,m})^{2D}}{||(\mathbf{W}^{(i)}_{j,\,:})^{2D}||_2}, & \text{if } ||(\mathbf{W}^{(i)}_{j,\,:})^{2D}||_2 > \lambda, \\ 0, & otherwise. \end{cases} \tag{7}$$

Because we train the network from scratch, to avoid excessive restraint in the early training phase, we introduce a penalty coefficient λ with a gradually increased value (i.e. from zero to a proper value). In particular, for simplicity, we use the sigmoid function to reflect the value of λ at the t-th iteration and $\lambda(t)$ is updated by

$$\lambda(t) = \frac{\lambda_{max}}{1 + e^{-(\frac{30t}{T} - 15)}}. \tag{8}$$

where λ_{max} is the maximum of the penalty coefficient and T is total iterations.

Dynamic Sparsity Algorithm

Channel Sensitivity Criteria. To identify the unimportant parameters set $\mathbf{\Theta^P}$, it is critical to calculate the sensitivity of channels. From the perspective of the loss function, the sensitivity of parameter w can be represented by

$$S(w) = |\mathcal{L}(\mathcal{F}(\mathbf{\Theta}_{w\to 0}, \mathbf{x}), \mathbf{y}) - \mathcal{L}(\mathcal{F}(\mathbf{\Theta}_w, \mathbf{x}), \mathbf{y})| \tag{9}$$

where $\mathbf{\Theta}_{w\to 0}$ denotes that parameter w tends to 0.

Like (Ding et al. 2019b), using Taylor series, the loss function can be represented

$$\mathcal{L}\left(\mathcal{F}(\mathbf{\Theta}_w, \mathbf{x}), \mathbf{y}\right) = \mathcal{L}(\mathcal{F}(\mathbf{\Theta}_{w\to 0}, \mathbf{x}), \mathbf{y}) - \frac{\partial \mathcal{L}\left(\mathcal{F}(\mathbf{\Theta}, \mathbf{x}), \mathbf{y}\right)}{\partial w} w + o\left(w^2\right) \tag{10}$$

where $o\left(w^2\right)$ is a remainder term. Because w is very small, $o\left(w^2\right)$ can be ignored.

Filter pruning removes filter-wise parameters and not just a single parameter, so the sensitivity of channel j concerning the current layer in the i-th layer is

$$\begin{aligned} &S_c((\mathbf{W}^{(i)}_{j,\,:})^{2D}_{out}) \\ &= |\mathcal{L}(\mathcal{F}(\mathbf{\Theta}_{(\mathbf{W}^{(i)}_{j,\,:})^{2D}_{out}\to\mathbf{0}}, \mathbf{x}), \mathbf{y}) - \mathcal{L}(\mathbf{\Theta}_{(\mathbf{W}^{(i)}_{j,\,:})^{2D}_{out}}, \mathbf{x}), \mathbf{y})| \\ &\approx |\frac{\partial \mathcal{L}\left(\mathcal{F}(\mathbf{\Theta}, \mathbf{x}), \mathbf{y}\right)}{\partial (\mathbf{W}^{(i)}_{j,\,:})^{2D}_{out}} (\mathbf{W}^{(i)}_{j,\,:})^{2D}_{out}| \\ &= |\sum_{m=1}^{r^{(i)}k_1^{(i)}k_2^{(i)}} \frac{\partial \mathcal{L}\left(\mathcal{F}(\mathbf{\Theta}, \mathbf{x}), \mathbf{y}\right)}{\partial (\mathbf{W}^{(i)}_{j,\,m})^{2D}_{out}} (\mathbf{W}^{(i)}_{j,\,m})^{2D}_{out}| \end{aligned} \tag{11}$$

While the channels are removed in the current layer, the corresponding input channels are pruned in the next layer (Li et al. 2019). So, the sensitivity concerning the next layer is

$$S_n((\mathbf{W}^{(i+1)}_{j,\,:})^{2D}_{in}) \approx |\sum_{m=1}^{c^{(i+1)}k_1^{(i+1)}k_2^{(i+1)}} \frac{\partial \mathcal{L}\left(\mathcal{F}(\mathbf{\Theta}, \mathbf{x}), \mathbf{y}\right)}{\partial (\mathbf{W}^{(i+1)}_{j,\,m})^{2D}_{in}} (\mathbf{W}^{(i+1)}_{j,\,m})^{2D}_{in}| \tag{12}$$

Algorithm 1: The Proposed Method.

```
   Input: $Network$, Dataset $\mathcal{D}$, $\lambda_{max}$, $\alpha$, pruning ratio $PR_0$.
1  $e = 0$;
2  For each layer $i$, initialize the sparsity allocation ratio $sr^{(i)}$;
3  while $e < Epoch$ do
4      for each iteration $\in$ epoch e do
5          Update: $\mathbf{\Theta} \leftarrow \mathbf{\Theta} - \alpha\nabla\mathcal{L}(\mathbf{\Theta})$ using dataset $\mathcal{D}$;
6          For each layer $i$, calculate the sensitivity set of channels: $S^{(i)}_{channel} = \{S_t(\mathcal{K}^{(i)}_1), S_t(\mathcal{K}^{(i)}_2), ..., S_t(\mathcal{K}^{(i)}_{c^{(i)}})\}$ via Equation (13);
7          For each layer $i$, sort and truncate the smallest-$\lceil sr^{(i)} * c^{(i)}\rceil$ of $S^{(i)}_{channel}$ to identify $\mathbf{\Theta^P}$ and $index$;
8          Update $\lambda$ via Equation (8);
9          for each $(W^{(i)})^{2D}[index^{(i)}] \in \mathbf{\Theta^P}$ do
10             Update $(\mathbf{W}^{(i)})^{2D}[index^{(i)}]$ via Equation (7);
11         end
12     end
13     For each layer $i$, recalculate $sr^{(i)}$;
14     $e = e + 1$;
15 end
16 Prune the redundant filters ($\left\|\mathbf{W}^{(i)}_{j,\,:}\right\|_2 = 0$) and return the compressed network with acceptable accuracy.
```

We consider the impact of the current and next layers and the total sensitivity of channel j in the i-th layer $S_t(\mathcal{W}^{(i)}_j)$ is

$$\begin{aligned} S_t(\mathcal{W}^{(i)}_j) = |&\sum_{m=1}^{r^{(i)}k_1^{(i)}k_2^{(i)}} \frac{\partial \mathcal{L}\left(\mathcal{F}(\mathbf{\Theta}, \mathbf{x}), \mathbf{y}\right)}{\partial (\mathbf{W}^{(i)}_{j,\,m})^{2D}_{out}} (\mathbf{W}^{(i)}_{j,\,m})^{2D}_{out} \\ &+ \sum_{m=1}^{c^{(i+1)}k_1^{(i+1)}k_2^{(i+1)}} \frac{\partial \mathcal{L}\left(\mathcal{F}(\mathbf{\Theta}, \mathbf{x}), \mathbf{y}\right)}{\partial (\mathbf{W}^{(i+1)}_{j,\,m})^{2D}_{in}} (\mathbf{W}^{(i+1)}_{j,\,m})^{2D}_{in}| \end{aligned} \tag{13}$$

Dynamic Sparsity Ratio Allocation. To adaptively learn a structured sparsity network, our method dynamically updates sparsity allocation ratios in different layers. Specifically, at the end of every epoch, We leverage the sparsity result from the epoch to recalculate the sparsity allocation ratio of every layer in the subsequent epoch. We denote relative sparsity allocation ratio in the i-th layer as $sr^{(i)}$, sparsity ratio (zero parameters ratio) as $sr^{(i)}_{zero}$, and sparsity-inducing ratio (the redundant non-zero parameters ratio) as $sr^{(i)}_{inducing}$. To avoid introducing extra layer-wise parameters, we use a uniform ratio to calculate the sparsity-inducing ratio across all layers. Then, sparsity allocation ratios are equal to sparsity ratios and sparsity-inducing ratios. During training, as sparsity ratios of different layers continuing to change, our sparsity allocation ratios are dynamically up-

Model Architecture	Method	Training Settings			Test Accuracy	PR	
		Pre-defined?	Pre-trained?	Fine-tuning?		$Params$	$FLOPs$
VGG-Small	Baseline	N/A	N/A	N/A	(93.85±0.07)%	0	0
	Li *et al.* (Li et al. 2016)	✔	✔	✔	93.40%	63.90%	34.21%
	NRE (Jiang et al. 2018)	✗	✔	✔	93.40%	92.72%	67.64%
	Slimming (Liu et al. 2017)	✗	✗	✔	93.48%	86.65%	43.50%
	HRank (Lin et al. 2020a)	✗	✔	✔	93.43%	82.90%	53.50%
	SFP (He et al. 2018a)	✔	✗	✗	92.66%	69.17%	69.24%
	SSS (Huang and Wang 2018)	✗	✗	✗	93.20%	66.67%	69.70%
	Ours	✗	✗	✗	**(93.52±0.15)%**	**93.32%**	**70.85%**
ResNet56	Baseline	N/A	N/A	N/A	(93.81±0.14%)	0	0
	Li *et al.* (Li et al. 2016)	✔	✔	✔	93.06%	13.70%	27.60%
	CP (He, Zhang, and Sun 2017)	✔	✔	✔	91.80%	--	50.00%
	AMC (He et al. 2018b)	✗	✔	✔	91.90%	--	50.00%
	HRank (Lin et al. 2020a)	✗	✔	✔	93.17%	42.40%	50.00%
	SFP (He et al. 2018a)	✔	✗	✗	(92.26±0.31)%	--	52.60%
	FPGM (He et al. 2019)	✔	✗	✗	(92.93±0.49)%	--	52.60%
	ASFP (He et al. 2020)	✔	✗	✗	(92.44±0.07)%	--	52.60%
	Ours	✗	✗	✗	**(93.20±0.11)%**	**46.84%**	**52.86%**

Table 1: The pruning results of VGG-Small and ResNet56 on CIFAR-10. The "Pre-defined?" refers to whether a fixed pruning ratio is pre-set in each layer through empirical studies. The "--" indicates that the results are not listed in the original paper.

dated. So, our method does not require pre-define the pruning architecture and can adaptively learn an optimal structured sparsity network.

Computational Complexity Analysis. In our proposed method, additional computation mainly comes from identifying and updating the expected pruning parameters. Because the gradients of parameters are obtained by a backward process in the SGD algorithm and updating the parameters are achieved by ISTA, the additional time consumption of DPFPS is acceptable. The detailed procedure of DPFPS is presented in Algorithm 1.

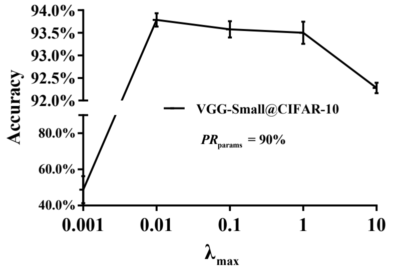


Figure 3: Test accuracy in different λ_{max} settings.

Experiments

Experimental Setup

1) Datasets and Networks: We evaluate DPFPS on two datasets: CIFAR (Krizhevsky and Hinton 2009) and ImageNet (Russakovsky et al. 2015). The same data augmentation strategies are used as PyTorch official examples (Paszke et al. 2017). On CIFAR-10, we evaluate the proposed method using VGG-16 (Simonyan and Zisserman 2014) and ResNet56 (He et al. 2016). As the original VGG-16 is specially designed for ImageNet classification, we use a variation version (i.e. VGG-Small) taken from (Zagoruyko 2015) in our experiment. On ImageNet dataset, we evaluate DPFPS on ResNets (including ResNet 34, 50, and 101) and MobileNet v2 (Sandler et al. 2018).

2) Implementation Details: All networks are trained from scratch. It takes 200 and 100 epochs on CIFAR-10 and ImageNet datasets, with an initial learning rate of 0.1, and a mini-batch size of 64 and 256, respectively. The learning rate is multiplied by 0.1 at 50% and 75% of the training epochs on CIFAR-10, and at 30, 60, and 90 epoch on ImageNet. We utilize an SGD optimizer with a weight decay of 10^{-4} and a momentum of 0.9. For MobileNet v2 on ImageNet, we use the settings like AMC (He et al. 2018b). All experiments are implemented on multiple NVIDIA RTX 2080 Ti GPUs and Intel(R) Xeon(R) Gold 5118 CPU by PyTorch. On CIFAR-10, considering fluctuations caused by different random seeds in the experimental results, we use the mean and standard deviation to report the results by conducting the same experiment 5 times.

3) Parameter Settings: In our experiments, the parameter λ_{max} impacts the structured sparsity results, so we study the performances in different λ_{max} settings. As shown in Figure 3, we tune λ_{max} exponentially in a relatively wide range (i.e. [0.001, 10]). When λ_{max} is set to be a small value, the test accuracy is low. This is because inadequate sparsity produces low precision after pruning the network, like (Wen et al. 2016; Liu et al. 2017), leading to needing fine-tuning the pruned network to recover the precision. When λ_{max} is set to be a big value, excessive sparsity causes the network parameters not to be learned well. In our experiment deployment, the parameter $\lambda_{max} = 0.01$ can meet the experiment results. For simplicity, we set the parameter λ_{max} to 0.01 in all experiments, though we can obtain better results by

Model Architecture	Method	Training Settings			Test Accuracy		PR_{FLOPs}
		Pre-defined?	Pre-trained?	Fine-tuning?	Top-1	Top-5	
ResNet34	Baseline	N/A	N/A	N/A	73.92%	91.62%	0
	Li *et al.* (Li et al. 2016)	✔	✔	✔	72.17%	−−	24.20%
	SFP (He et al. 2018a)	✔	✗	✗	71.83%	90.33%	41.10%
	FPGM (He et al. 2019)	✔	✗	✗	72.11%	90.69%	41.10%
	ASFP (He et al. 2020)	✔	✗	✗	71.72%	90.65%	41.10%
	Ours	✗	✗	✗	**72.25%**	**90.80%**	**43.29%**
ResNet50	Baseline	N/A	N/A	N/A	76.15%	92.87%	0
	ThiNet (Luo et al. 2019)	✔	✔	✔	74.03%	92.11%	36.80%
	CP (He, Zhang, and Sun 2017)	✔	✔	✔	−−	90.80%	**50.00%**
	HRank (Lin et al. 2020a)	✗	✔	✔	74.98%	92.33%	43.77%
	SFP (He et al. 2018a)	✔	✗	✗	74.61%	92.06%	41.80%
	FPGM (He et al. 2019)	✔	✗	✗	75.03%	92.40%	42.20%
	ASFP (He et al. 2020)	✔	✗	✗	74.88%	92.39%	41.80%
	Ours	✗	✗	✗	**75.55%**	**92.54%**	46.20%
ResNet101	Baseline	N/A	N/A	N/A	77.37%	93.56%	0
	SFP (He et al. 2018a)	✔	✗	✗	77.03%	93.46%	42.20%
	Ours	✗	✗	✗	**77.27%**	**93.68%**	**44.97%**
MobileNet v2	Baseline	N/A	N/A	N/A	72.00%	−−	0
	0.75x MobileNet v2 (Sandler et al. 2018)	✗	✗	✗	69.80%	−−	**26.54%**
	AMC (He et al. 2018b)	✗	✔	✔	70.80%	−−	**26.54%**
	Ours	✗	✗	✗	**71.10%**	−−	24.89%

Table 2: The pruning results of ResNets and MobileNet v2 on ImageNet.

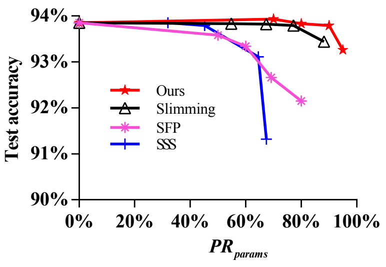


Figure 4: Performance comparison with different baselines.

fine-tuning λ_{max} in different experiments.

4) Evaluation Metrics: To evaluate the performance of DPFPS, we use the parameters or FLOPs pruning ratio:

$$PR_{Params/FLOPs} = 1 - \frac{\#Pruned\ Params/FLOPs}{\#Original\ Params/FLOPs} \tag{14}$$

where "$\#Pruned\ Params/FLOPs$" denote the remaining parameters or FLOPs after the model is pruned, respectively.

Performance Comparison with Different Baselines

We use VGG-Small on CIFAR-10 and compare DPFPS with three baselines: SFP, SSS and Slimming. As shown in Figure 4, when PR_{Params} is less than 50%, SFP is inferior to other methods. This is because SFP adopts the fixed pruning structure and does not learn an optimal pruning network. With the increase of PR_{Params}, for SSS, the larger penalty needs to be imposed upon the parameters. When PR_{Params} reaches 65%, the test accuracy of SSS sharply reduces. So, for the need of a large pruning ratio, due to excessive restraint, SSS is inappropriate and fine-tuning is needed to recover the precision like Slimming. In all the baselines, DPFPS achieves the best trade-off curve between test accuracy and pruning ratio, exceeding the fine-tuning method (i.e. Slimming). These show that our dynamic and progressive structured sparsity regularization is effective. More importantly, our proposed method alleviates the cumbersome pruning process, without any pre-trained, fine-tuning, or multi-pass scheme, which is efficient.

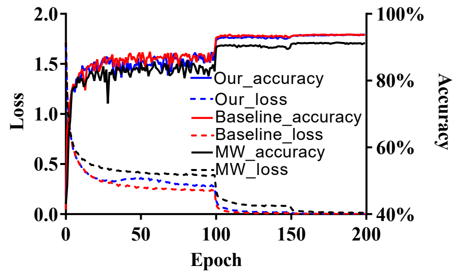
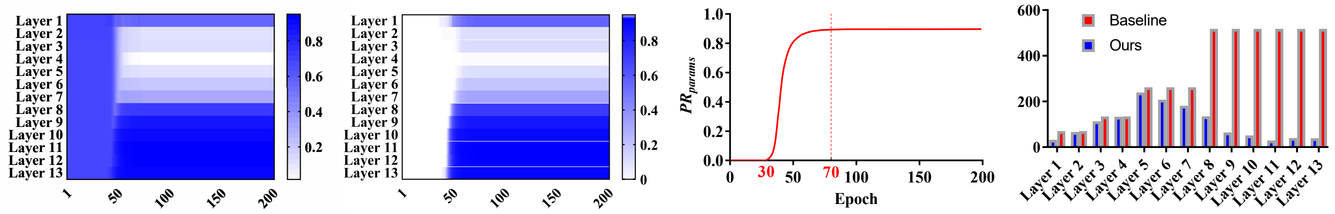


Figure 5: Analysis of training convergence.

Performance Comparison with State-of-the-Art Methods

We also compare DPFPS with 11 state-of-the-arts.

On CIFAR-10, we compare our method with Li *et al.*, NRE, Slimming, SFP, CP, AMC, FPGM, SSS, ASFP, and HRank, shown in Table 1. For VGG-Small, the proposed method achieves an accuracy of 93.52% with 93.32% pruning ratio of parameters and 70.85% pruning ratio of FLOPs,



(a) Sparsity allocation ratio. (b) Sparsity ratio. (c) Sparsity ratio of the network. (d) Channel number.

Figure 6: Analysis of visualization in the different layers and network.

Method	VGG-Small@CIFAR-10		ResNet34@ImageNet	
	Accuracy	PR_{FLOPs}	Top-1	PR_{FLOPs}
Ours(no)	93.01%	76.26%	72.22%	38.88%
Ours	**93.26%**	**77.43%**	**72.25%**	**43.29%**

Table 3: The comparison results with and without the progressive penalty.

much better than all the competing methods. Compared with the baseline, our method prunes VGG-Small more than 93% parameters and 70% FLOPs, with a slight accuracy loss. For more compact architecture ResNet56, our method prunes more than 52% pruning ratio of FLOPs and outperforms other methods in performance.

On ImageNet, our method is compared with Li *et al.*, ThiNet, CP, SFP, FPGM, ASFP, AMC, and HRank. The results are reported in Table 2. For ResNet34, the proposed method achieves the best performance. For ResNet50, our method obtains the highest accuracy, with a second highest pruning ratio of FLOPs (46.21% vs 50.00%). For a deeper network of ResNet101, we obtain the highest pruning ratio, even with an increase of 0.12% Top-5 accuracy. For lightweight network compression (i.e., MobileNet v2), our method also obtains the best accuracy. Thus, it can be concluded that the proposed method is superior to all the competing methods, due to introducing the dynamic and progressive structured sparsity regularization.

Analysis and Discussion

Training Convergence Analysis. In terms of training convergence, Figure 5 plots the training loss and test accuracy curves of our proposed method, WM, and baseline on VGG-Small@CIFAR-10. To achieve thinner models, WM is used to uniformly prune the channels of each layer, such as MobileNet. We set $PR_{Params} = 90\%$ in our method and WM. Due to our regularization with a gradually increased value, at early training epochs, our training loss and test accuracy are consistent with the original network, better than WM. As the penalty continues to increase, our training loss and test accuracy are slightly worse than the baseline but better than WM. But after 100 epochs, our method achieves nearly the same training loss and test accuracy as the baseline and better than WM. It can be concluded that the proposed method has comparable convergence speed and final performance with the baseline and better than WM.

Efficacy Analysis. To evaluate the effectiveness of our method, on one hand, we analyze the sparsity ratio over

Dataset	Method	$FLOPs$ (Backbone)	mAP
PASCAL VOC2007	Faster R-CNN	4.08G	0.734
	Ours	2.81G	0.726

Table 4: Results on PASCAL VOC 2007 dataset.

epoch and pruned structure on VGG-Small@CIFAR-10. As shown in Figure 6(a) and 6(b), during training, our method dynamically allocates the sparsity ratio to each layer and adaptively learns the sparsity of each layer in the network, which is beneficial to learn a structured sparsity network with a good performance, as shown in Figure 6(c). Additionally, we visualize the number of each layer between our method and baseline in Figure 6(d), which shows that deep layers have more redundant than shallow layers. These results show our method is effective due to our dynamic structured sparsity regularization. On the other hand, we also analyze the effectiveness of the progressive penalty. As shown in Table 3, our method is superior to the regularization without the progressive penalty. These results show our method is effective due to the progressive structured sparsity regularization.

Its Application to Detection Tasks. As described above, our method is effective in image classification. To further analyze the generalization of our method, we use ResNet50 as a backbone network to deploy Faster R-CNN (Ren et al. 2015) for object detection, and then compress Faster R-CNN by reducing 30% FLOPs of the backbone network. In the experimental implementation, we evaluate the performance with mean Average Precision (mAP) on PASCAL VOC 2007 (Everingham et al. 2015) dataset. As shown in Table 4, our pruned model shows a good result. The mAP of our method is slightly lower than the baseline, but the inference speed is faster than the baseline. This demonstrates that our method has good generalization on other tasks.

Conclusion

In this paper, we have proposed a novel Dynamic and Progressive Filter Pruning scheme (DPFPS) that directly learns a structured sparsity network from Scratch. It has imposed a new structured sparsity-inducing regularization specifically upon the expected pruning parameters in a dynamic sparsity manner. Moreover, We have designed a group Lasso based progressive penalty regularization, making the sparsity process to be soft and alleviating the harm on model performance. Extensive experiments have shown that our method is competitive with state-of-the-art methods.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China (Grant No. 2020AAA0106800, No. 2018AAA0102802, No.2018AAA0102803, and No. 2018AAA0102800), in part by the Natural Science Foundation of China (Grant No. 62036011, No. 61902401, No. 61972071, No. 61906052, No. 61772225, No. 61721004, No. U1803119, No. U1736106, and No. 6187610), in part by the NSFC-General Technology Collaborative Fund for Basic Research (Grant No. U1936204), in part by the Beijing Natural Science Foundation (Grant No. JQ18018), in part by the Key Research Program of Frontier Sciences, CAS, Grant No. QYZDJ-SSW-JSC040, in part by the National Natural Science Foundation of Guangdong (No. 2018B030311046), and in part by the CAS External cooperation key project. The work of Bing Li was also supported by the Youth Innovation Promotion Association, CAS.

References

Alvarez, J. M.; and Salzmann, M. 2016. Learning the Number of Neurons in Deep Networks. In *Advances in Neural Information Processing Systems*, 2270–2278.

Beck, A.; and Teboulle, M. 2009. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences* 2(1): 183–202.

Bertinetto, L.; Valmadre, J.; Henriques, J. F.; Vedaldi, A.; and Torr, P. H. 2016. Fully-convolutional siamese networks for object tracking. In *Proceedings of European Conference on Computer Vision*, 850–865.

Boyd, S.; and Vandenberghe, L. 2004. *Convex optimization*. Cambridge university press.

Ding, X.; Ding, G.; Guo, Y.; Han, J.; and Yan, C. 2019a. Approximated Oracle Filter Pruning for Destructive CNN Width Optimization. In *Proceedings of International Conference on Machine Learning*, 1607–1616.

Ding, X.; Ding, G.; Han, J.; and Tang, S. 2018. Auto-balanced Filter Pruning for Efficient Convolutional Neural Networks. In *Proceedings of AAAI Conference on Artificial Intelligence*, 6797–6804.

Ding, X.; Ding, G.; Zhou, X.; Guo, Y.; Han, J.; and Liu, J. 2019b. Global Sparse Momentum SGD for Pruning Very Deep Neural Networks. In *Advances in Neural Information Processing Systems*, 6382–6394.

Everingham, M.; Eslami, S. M.; Van Gool, L.; Williams, C. K. I.; Winn, J.; and Zisserman, A. 2015. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision* 111(1): 98–136.

Girshick, R.; Donahue, J.; Darrell, T.; and Malik, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 580–587.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 1135–1143.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

He, Y.; Dong, X.; Kang, G.; Fu, Y.; Yan, C.; and Yang, Y. 2020. Asymptotic Soft Filter Pruning for Deep Convolutional Neural Networks. *IEEE transactions on Cybernetics* 50(8): 3594 – 3604.

He, Y.; Kang, G.; Dong, X.; Fu, Y.; and Yang, Y. 2018a. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2234–2240.

He, Y.; Lin, J.; Liu, Z.; Wang, H.; Li, L.-J.; and Han, S. 2018b. AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In *Proceedings of European Conference on Computer Vision*, 815–832.

He, Y.; Liu, P.; Wang, Z.; Hu, Z.; and Yang, Y. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4340–4349.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 1389–1397.

Huang, Z.; and Wang, N. 2018. Data-Driven Sparse Structure Selection for Deep Neural Networks. In *Proceedings of European Conference on Computer Vision*, 317–334.

Idelbayev, Y.; and Carreira-Perpinan, M. A. 2020. Low-Rank Compression of Neural Nets: Learning the Rank of Each Layer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8049–8059.

Jiang, C.; Li, G.; Qian, C.; and Tang, K. 2018. Efficient DNN Neuron Pruning by Minimizing Layer-wise Nonlinear Reconstruction Error. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2298–2304.

Jin, Q.; Yang, L.; and Liao, Z. 2020. AdaBits: Neural Network Quantization With Adaptive Bit-Widths. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2146–2156.

Krizhevsky, A.; and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 1097–1105.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* .

Li, J.; Qi, Q.; Wang, J.; Ge, C.; Li, Y.; Yue, Z.; and Sun, H. 2019. OICSR: Out-In-Channel Sparsity Regularization for Compact Deep Neural Networks. In *Proceedings of the*

IEEE Conference on Computer Vision and Pattern Recognition, 7046–7055.

Lin, M.; Ji, R.; Wang, Y.; Zhang, Y.; Zhang, B.; Tian, Y.; and Shao, L. 2020a. HRank: Filter Pruning Using High-Rank Feature Map. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1529–1538.

Lin, S.; Ji, R.; Li, Y.; Deng, C.; and Li, X. 2020b. Toward Compact ConvNets via Structure-Sparsity Regularized Filter Pruning. *IEEE Transactions on Neural Networks and Learning Systems* 31(2): 574–588.

Liu, N.; Ma, X.; Xu, Z.; Wang, Y.; Tang, J.; and Ye, J. 2020. AutoCompress: An Automatic DNN Structured Pruning Framework for Ultra-High Compression Rates. In *Proceedings of AAAI Conference on Artificial Intelligence*, 4876–4883.

Liu, Y.; Cao, J.; Li, B.; Yuan, C.; Hu, W.; Li, Y.; and Duan, Y. 2019a. Knowledge Distillation via Instance Relationship Graph. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7096–7104.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, 2736–2744.

Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; and Darrell, T. 2019b. Rethinking the Value of Network Pruning. In *International Conference on Learning Representations*.

Luo, J.; Zhang, H.; Zhou, H.; Xie, C.; Wu, J.; and Lin, W. 2019. ThiNet: Pruning CNN Filters for a Thinner Net. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41(10): 2525–2538.

Luo, J.-H.; Wu, J.; and Lin, W. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, 5058–5066.

Molchanov, P.; Mallya, A.; Tyree, S.; Frosio, I.; and Kautz, J. 2019. Importance Estimation for Neural Network Pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 11264–11272.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems Workshop*.

Peng, H.; Wu, J.; Chen, S.; and Huang, J. 2019. Collaborative Channel Pruning for Deep Networks. In *Proceedings of the International Conference on Machine Learning*, 5113–5122.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, 91–99.

Ruan, X.; Liu, Y.; Yuan, C.; Li, B.; Hu, W.; Li, Y.; and Maybank, S. 2020. EDP: An Efficient Decomposition and Pruning Scheme for Convolutional Neural Network Compression. *IEEE Transactions on Neural Networks and Learning Systems* 1–15.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115(3): 211–252.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–4520.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* .

Singh, P.; Verma, V. K.; Rai, P.; and Namboodiri, V. P. 2019. Play and Prune: Adaptive filter pruning for deep model compression. In *Proceedings of International Joint Conference on Artificial Intelligence*, 3460–3466.

Venugopalan, S.; Rohrbach, M.; Donahue, J.; Mooney, R.; Darrell, T.; and Saenko, K. 2015. Sequence to sequence-video to text. In *Proceedings of the IEEE International Conference on Computer Vision*, 4534–4542.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, 2074–2082.

Yu, R.; Li, A.; Chen, C.-F.; Lai, J.-H.; Morariu, V. I.; Han, X.; Gao, M.; Lin, C.-Y.; and Davis, L. S. 2018. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9194–9203.

Yuan, M.; and Lin, Y. 2006. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68(1): 49–67.

Zagoruyko, S. 2015. 92.45% on CIFAR-10 in Torch. *Torch Blog* .

Zhang, T.; Ye, S.; Zhang, K.; Tang, J.; Wen, W.; Fardad, M.; and Wang, Y. 2018. A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers. In *Proceedings of European Conference on Computer Vision*, 184–199.