Automatic Neural Network Pruning that Efficiently Preserves the Model Accuracy

Thibault Castells and Seul-Ki Yeom*

Nota AI GmbH
Friedrichstrasse 200, 10117 Berlin, Germany
thibault@nota.ai, skyeom@nota.ai

Abstract

Neural networks performance has been significantly improved in the last few years, at the cost of an increasing number of floating point operations (FLOPs). When computational resources are limited, more FLOPs becomes an issue. As an attempt to solve this problem, pruning filters is a common solution, but most existing pruning methods do not preserve the model accuracy efficiently and therefore require a large number of finetuning epochs. In this paper, we propose an automatic pruning method that learns which neurons to preserve in order to maintain the model accuracy while reducing the FLOPs to a predefined target. To accomplish this task, we introduce a trainable bottleneck that only requires 25.6% (CIFAR-10) or 15.0% (ILSVRC2012) of the dataset within one single epoch to learn which filters to prune. Experiments on various architectures and datasets show that the proposed method can not only preserve the accuracy after pruning but also outperform existing methods after finetuning. With 52.00% FLOPs reduction on ResNet-50, we achieve a Top-1 accuracy of 47.51% after pruning and a state-of-the-art (SOTA) accuracy of 76.63% after finetuning on ILSVRC2012. Code available at https://github.com/nota-github/autobot_AAAI23.

1 Introduction

In the last decade, Deep Neural Networks (DNNs) popularity has grown exponentially as the results improved, and they are now used in a variety of applications such as classification, detection, etc. However, these improvements are often faced with increasing model complexity, resulting in a need for more computational resources. Various attempts to make heavy models more compact have been proposed, based on different compression methods such as knowledge distillation (Polino, Pascanu, and Alistarh 2018; Guo et al. 2020), pruning (Li et al. 2017; Lin et al. 2020a,b; Yeom et al. 2021), quantization (Qu et al. 2020), neural architecture search (NAS) (Yang et al. 2021), *etc*. Network pruning, which consists in removing redundant and unimportant connections, received great interest from the industry as it is a simple and effective solution. While the main challenge of this method is to find a good pruning criterion, another difficulty is to define what percentage of each layer should be pruned. As a manual search is a time-consuming process that requires human expertise, recent works have proposed methods that automatically prune the redundant filters throughout the network to meet a given constraint such as the number of parameters, FLOPs, or hardware platform (Liu et al. 2017; You et al. 2019; Li et al. 2021; Molchanov et al. 2019; Lin et al. 2020b; Yeom et al. 2021; Yu et al. 2018; Dai et al. 2018; Zheng et al. 2021). To automatically find the best-pruned architectures, these methods rely on various metrics such as the 2nd order Taylor expansions (Molchanov et al. 2019), the layer-wise relevance propagation score (Yeom et al. 2021), *etc*. For further details, please find Sec. 2. Although these strategies improved over time, they usually do not explicitly aim to preserve the model accuracy, or they do it in a computationally expensive way.

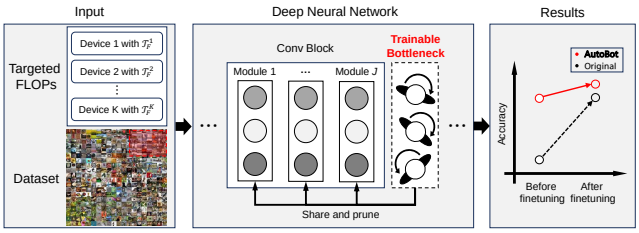

Figure 1: System flow of AutoBot for automatic network pruning. A trainable bottleneck is injected after each convolution block. They are then updated to restrict the information flow (like water taps) while minimizing the accuracy drop, with the given targeted FLOPs and a small amount of data. The most restricted filters are pruned afterwards. As a result, compared to other existing pruning methods, AutoBot can efficiently preserve the accuracy, leading to a SOTA accuracy after finetuning.

In this paper, we make the hypothesis that, for the same compression, the pruned architecture that can lead to the best accuracy after finetuning is the one that most efficiently preserves the accuracy during the pruning process (see Sec. 4.5). We therefore introduce an automatic pruning method, called AutoBot, that uses trainable bottlenecks to efficiently preserve the model accuracy while minimizing the FLOPs, as shown in Fig. 1. These bottlenecks only require one single epoch of training with 25.6% (CIFAR-10)

*Corresponding author.

or 15.0% (ILSVRC2012) of the dataset to efficiently learn which filters to prune. We compare AutoBot with various pruning methods, and show a significant improvement of the pruned models before finetuning, leading to a SOTA accuracy after finetuning. We also perform a practical deployment test on several edge devices to demonstrate the speed improvement of the pruned models.

To summarize, our contributions are as follows:

- We introduce AutoBot, a novel automatic pruning method that uses a trainable bottleneck to efficiently learn which filter to prune in order to maximize the accuracy while minimizing the FLOPs of the model. This method can easily and intuitively be implemented regardless of the dataset or model architecture.
- We demonstrate that preserving the accuracy during the pruning process has a strong impact on the accuracy of the finetuned model (Sec. 4.5).
- Extensive experiments show that AutoBot efficiently preserve the accuracy after pruning (before finetuning), and outperforms previous pruning methods once finetuned.

2 Related Works

In this section, we summarize some related works compared to our proposed method. Traditionally, magnitude-based pruning aims to exploit the inherent characteristics of the network to define a pruning criterion, without modifying the network parameters. Popular criteria include l_p-norm (Li et al. 2017; Lin et al. 2021; Li et al. 2020), Taylor expansion (Molchanov et al. 2019), Gradient (Liu and Wu 2019), Singular Value Decomposition (Lin et al. 2020a), sparsity of output feature maps (Hu et al. 2016), geometric median (He et al. 2019), etc. Recently, Tang et al. (2020) proposed a scientific control pruning method, called SCOP, which introduces knockoff features as the control group. In contrast, adaptive pruning needs to retrain the networks from scratch with a modified training loss or architecture which adds new constraints. Several works (Liu et al. 2017; Luo, Wu, and Lin 2017; Ye et al. 2018) add trainable parameters to each feature map channel to obtain data-driven channel sparsity, enabling the model to automatically identify redundant filters. Luo, Wu, and Lin (2017) introduce Thinet that formally establishes filter pruning as an optimization problem and prunes filters based on statistical information computed from its next layer, not the current layer. Lin et al. (2019) propose a structured pruning method that jointly prunes filters and other structures by introducing a soft mask with sparsity regularization. However, retraining the model from scratch is a time- and resource-consuming process that does not significantly improve the accuracy compared to magnitude-based pruning. Although these two pruning strategies are intuitive, the pruning ratio must be manually defined layer-by-layer, which is a time-consuming process that requires human expertise. Instead, in this paper, we focus on automatic pruning.

As suggested by the name, automatic network pruning removes the redundant filters throughout the network automatically under any constraints such as a number of parameters, FLOPs, or hardware platform. In this respect, a large number of automatic pruning methods have been proposed. Liu et al. (2017) optimize the scaling factor γ in the batch-norm layer as a channel selection indicator to decide which channels are unimportant. You et al. (2019) propose an automatic pruning method, called Gate Decorator, which transforms CNN modules by multiplying their output by channel-wise scaling factors and adopt an iterative pruning framework called Tick-Tock to boost pruning accuracy. Li et al. (2021) propose a collaborative compression method that mutually combines channel pruning and tensor decomposition. Molchanov et al. (2019) estimates the contribution of a filter to the final loss using 2nd order Taylor expansions and iteratively removes those with smaller scores. Lin et al. (2020b) propose ABCPruner to find the optimal pruned structure automatically by updating the structure set and recalculating the fitness. Back-propagation methods (Yeom et al. 2021; Yu et al. 2018) compute the relevance score of each filter by following the information flow from the model output. Dai et al. (2018) and Zheng et al. (2021) adopt information theory to preserve the information between the hidden representation and input or output.

Most existing methods are computationally and time expensive because they either require to retrain the model from scratch (Liu et al. 2017), apply iterative pruning (You et al. 2019; Molchanov et al. 2019; Yeom et al. 2021; Yu et al. 2018; Li et al. 2021) or finetune the model while pruning (Lin et al. 2020b; Dai et al. 2018). When the model isn't retrained or finetuned during the pruning process, they generally do not preserve the model accuracy after pruning (Zheng et al. 2021; Yeom et al. 2021; Yu et al. 2018), and thus require to be finetuned for a large number of epochs. In contrast to other automatic pruning methods, AutoBot stands out by its speed and its ability to preserve the accuracy of the model during the pruning process.

Algorithm 1: $AutoBot$

Input: pre-trained model f, targeted FLOPs F_T, acceptable FLOPs error ϵ, hyper-parameter β, number of iterations k, training data D
Output: Pruned model f'

Inject $Trainable\ Bottlenecks$ in f
for Batch $\mathcal{X}$ in $D[0;k]$ **do**
  $\mathcal{L} \leftarrow \mathcal{L}_{CE}(f(\mathcal{X};\mathbf{\Lambda})) + \beta\mathcal{L}_g(\mathbf{\Lambda})$
  $\mathbf{\Lambda} \leftarrow Update(\mathbf{\Lambda}, \mathcal{L})$
end for
$\mathbf{\Lambda}_{bool} \leftarrow GetPruningMask(\mathbf{\Lambda}, F_T, \epsilon)$
Remove $Trainable\ Bottlenecks$ from f
$f' \leftarrow Prune(f, \mathbf{\Lambda}_{bool})$
$f' \leftarrow Finetune(f', D)$
return f'

3 Method

Motivated by several bottleneck approaches (Tishby, Pereira, and Bialek 2000; Alemi et al. 2017; Schulz et al. 2020), our method control the information flow throughout the pretrained network using *Trainable Bottlenecks* that are injected into the model. The objective function of the train-

Algorithm 2: $GetPruningMask$

Input: trained bottlenecks parameters $\mathbf{\Lambda}$, targeted FLOPs F_T, acceptable FLOPs error ϵ
Output: pruning mask $\mathbf{\Lambda}_{bool}$

$\mathcal{T} \leftarrow 0.5$
$\mathbf{\Lambda}_{bool} \leftarrow 1$ where $\mathbf{\Lambda} > \mathcal{T}$, 0 elsewhere
$F \leftarrow \mathrm{g}(\mathbf{\Lambda}_{bool})$ (Eq. 5)
$i \leftarrow 0$
while $|F - F_T| > \epsilon$ **do**
  if $F > F_T$ **then**
    $\mathcal{T} \leftarrow \mathcal{T} + \frac{0.25}{2^i}$
  else
    $\mathcal{T} \leftarrow \mathcal{T} - \frac{0.25}{2^i}$
  end if
  $\mathbf{\Lambda}_{bool} \leftarrow 1$ where $\mathbf{\Lambda} > \mathcal{T}$, 0 elsewhere
  $F \leftarrow \mathrm{g}(\mathbf{\Lambda}_{bool})$
  $i \leftarrow i + 1$
end while
return $\mathbf{\Lambda}_{bool}$

able bottleneck is to maximize the information flow from input to output while minimizing the loss by adjusting the amount of information in the model under the given constraints. During the training procedure, only the parameters $\mathbf{\Lambda}$ of the trainable bottlenecks are updated while all the pretrained parameters of the model are frozen.

Compared to other pruning methods inspired by the information bottleneck (Dai et al. 2018; Zheng et al. 2021), we do not consider the compression of mutual information between the input/output and the hidden representations in order to evaluate the information flow. Such methods are orthogonal to AutoBot, which explicitly quantifies how much information is passed through each layer. This explicit quantification result in a faster training –we optimize the trainable bottlenecks on a fraction of one single epoch only– and an improved capacity to preserve the accuracy. Our AutoBot pruning process is summarized in Alg. 1.

3.1 Trainable Bottleneck

We formally define the concept of trainable bottleneck as an operator that can restrict the information flow throughout the network during the forward pass, using trainable parameters. Mathematically, it can be formulated as:

$$X_{i+1} = B(\boldsymbol{\lambda}_i, X_i) \tag{1}$$

where B stands for the trainable bottleneck, $\boldsymbol{\lambda}_i$ denotes the bottleneck parameters of the i^{th} operator, and X_i and X_{i+1} denote the input and output feature map of the bottleneck at the i^{th} operator, respectively. For instance, Schulz *et al*. (Schulz et al. 2020) control the amount of information into the model by injecting noise into it. In this case, B is expressed as $B(\boldsymbol{\lambda}_i, X_i) = \boldsymbol{\lambda}_i X_i + (1 - \boldsymbol{\lambda}_i)\epsilon$ where ϵ denotes the noise.

Inspired by the information bottleneck concept (Tishby, Pereira, and Bialek 2000; Alemi et al. 2017), we formulate a general bottleneck that is not limited to only information theory but can be optimized to satisfy any constraint as follow:

$$\min_{\mathbf{\Lambda}} \mathcal{L}_{CE}(\mathcal{Y}, f(\mathcal{X}; \mathbf{\Lambda})) \qquad s.t. \quad r(\mathbf{\Lambda}) \leq \mathcal{C} \tag{2}$$

where $\mathcal{L}_{CE}$ stands for the cross-entropy loss, $\mathcal{X}$ and $\mathcal{Y}$ stand for the model input and output, $\mathbf{\Lambda}$ is the set of the bottleneck parameters ($\mathbf{\Lambda} = [\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_L]$) in the model, r is a constraint function, and $\mathcal{C}$ is the desired constraint.

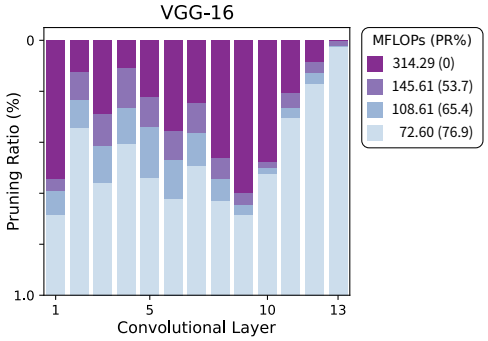

Figure 2: Per-layer filter pruning ratio for various targeted FLOPs on VGG-16. This ratio is automatically determined by AutoBot to satisfy the targeted FLOPs.

3.2 Pruning Strategy

In the following, we define a convolution block as a convolution layer, plus all the following operators that preserve the number and order of channels. It can contain multiple convolutions if their outputs are merged (in the case of a skip connection). In this work, we inject a bottleneck into each convolution block throughout the network such that the information flow of the estimated model to be pruned is quantified by restricting trainable parameters layer-wisely.

Compared to previous works, our bottleneck function $B(\boldsymbol{\lambda}_i, X_i)$ (Eq. 1) do not use noise to control the information flow:

$$X_{i+1} = \boldsymbol{\lambda}_i X_i \tag{3}$$

where $\boldsymbol{\lambda}_i \in [0, 1]$. Therefore the range of X_{i+1} is changing from $[\epsilon, X_i]$ to $[0, X_i]$. For pruning, this is more relevant since replacing an operator input by zeros is equivalent to pruning the operator (i.e. pruning the corresponding output of the previous operator).

Following the general objective function of the trainable bottleneck (Eq. 2), we introduce a regularizer g to constrain the FLOPs of the pruned architecture:

$$\min_{\mathbf{\Lambda}} \mathcal{L}_{CE}(\mathcal{Y}, f(\mathcal{X}; \mathbf{\Lambda})) \qquad s.t. \quad g(\mathbf{\Lambda}) = \mathcal{T}_F \tag{4}$$

where $\mathcal{T}_F$ is the target FLOPs (manually fixed), and $g(\mathbf{\Lambda})$ estimates the FLOPs of the model weighted by Λ. Formally, given a neural network consisting of multiple convolutional blocks, we define g as follows:

$$g(\mathbf{\Lambda}) = \sum_{i=1}^{L} \sum_{j=1}^{J_i} g_i^j(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{i-1}) \tag{5}$$

where $\boldsymbol{\lambda}_i$ is the vector of parameters of the information bottleneck following the i^{th} convolution block, g_i^j is the function that computes the FLOPs of the j^{th} operator of the i^{th} convolution block weighted by $\boldsymbol{\lambda}_i$, L is the total number of convolution blocks in the model and J_i is the total number of operators in the i^{th} convolution block. For instance, if g_i^j is for a convolutional operator without bias and padding, it is expressed as:

$$g_i^j(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{i-1}) = sum(\boldsymbol{\lambda}_i) \times sum(\boldsymbol{\lambda}_{i-1}) \times h \times w \times k \times k \quad (6)$$

where h and w are the height and width of the output feature map of the convolution, and k is its kernel size. Notice that within the i^{th} convolution block, all operators share $\boldsymbol{\lambda}_i$. That is, at a block level all the operators belonging to the same convolution block are pruned together.

To solve our optimization problem defined in Eq. 4, we introduce $\mathcal{L}_g$, a loss term designed to satisfy the constraint g from Eq. 5. We formulate $\mathcal{L}_g$ as follow:

$$\mathcal{L}_g = \begin{cases} \frac{g(\boldsymbol{\Lambda}) - \mathcal{T}_F}{\mathcal{M}_F - \mathcal{T}_F}, & \text{if } g(\boldsymbol{\Lambda}) \geq \mathcal{T}_F \\ 1 - \frac{g(\boldsymbol{\Lambda})}{\mathcal{T}_F}, & \text{otherwise} \end{cases} \quad (7)$$

where $\mathcal{M}_F$ is the FLOPs of the original model, and $\mathcal{T}_F$ is the predefined target FLOPs.

In contrast to g, this loss term is normalized such that the scale of the loss is always the same. As a result, for a given dataset, the training parameters are stable across different architectures. The optimization problem to update the proposed information bottlenecks for automatic pruning can be summarized as follows:

$$\min_{\boldsymbol{\Lambda}} \mathcal{L}_{CE}(\mathcal{Y}, f(\mathcal{X}; \boldsymbol{\Lambda})) + \beta \mathcal{L}_g(\boldsymbol{\Lambda}) \quad (8)$$

where β is a hyper-parameter that indicates the relative importance of its associated objective.

From $\boldsymbol{\Lambda}$ to pruning mask Once the bottlenecks are trained, $\boldsymbol{\Lambda}$ can be directly used as a pruning criterion. Therefore, we propose a way to quickly find the threshold under which neurons should be pruned. Since our bottleneck allows us to quickly and accurately compute the weighted FLOPs (Eq. 5), we can estimate the FLOPs of the model to be pruned without actual pruning. This is done by setting $\boldsymbol{\Lambda}$ to zero for the filters to be pruned, or one otherwise. We call this process *pseudo-pruning*. In order to find the optimal threshold, we initialize a threshold to 0.5 and *pseudo-prune* all filters with $\boldsymbol{\Lambda}$ lower than this threshold. We then compute the weighted FLOPs, and adopt the binary search algorithm to efficiently minimize the distance between the current and targeted FLOPs. This process is repeated until the gap is small enough. This process is summarized in Alg. 2. Once we have found the optimal threshold, we cut out all bottlenecks from the model and finally prune all the filters with $\boldsymbol{\Lambda}$ lower than this threshold to get the compressed model with the targeted FLOPs. This whole process takes less than a second on CPU as it is based on the binary search algorithm, which has a complexity of $\mathcal{O}(\log n)$, n being the number of FLOPs in this case.

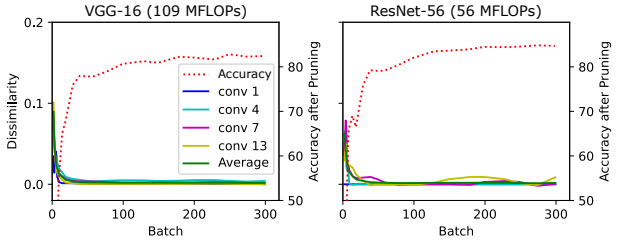

Figure 3: Evolution of accuracy after pruning (before fine-tuning) and of dissimilarity between filters ranking (normalised Kendall tau distance) when increasing the number of batches, on CIFAR-10.

Parametrization Following Schulz et al. (2020), we do not directly optimize $\boldsymbol{\Lambda}$ as this would require to use clipping to stay in the $[0, 1]$ interval. Instead, we parametrize $\boldsymbol{\Lambda} = \text{sigmoid}(\boldsymbol{\Psi})$, where the elements of $\boldsymbol{\Psi}$ are in $\mathbb{R}$.

Reduced training data We empirically observed that the training for the bottlenecks can converge quickly before the end of the first epoch. For instance, we can observe on Fig. 3 that around 200 batches are needed (25.6% of the dataset) to converge on CIFAR-10. For ILSVRC2012, the same observation is made with 15.0% of the dataset. Therefore, it suggests that regardless of model size (i.e. FLOPs), the optimally pruned architecture can be efficiently estimated using only a small portion of the dataset.

4 Experiments

4.1 Experimental Settings

To demonstrate the efficiency of AutoBot on a variety of experimental setups, experiments are conducted on two popular benchmark datasets and five common CNN architectures, 1) CIFAR-10 (Krizhevsky, Hinton et al. 2009) with VGG-16 (Simonyan and Zisserman 2015), ResNet-56/110 (He et al. 2016), DenseNet (Huang et al. 2017), and GoogLeNet (Szegedy et al. 2015), and 2) ILSVRC2012 (ImageNet) (Deng et al. 2009) with ResNet-50.

Experiments are performed within the *PyTorch* and *torchvision* frameworks (Paszke et al. 2017) under *Intel(R) Xeon(R) Silver 4210R CPU 2.40GHz* and *NVIDIA RTX 2080 Ti with 11GB* for GPU processing.

For CIFAR-10, we trained the bottlenecks for 200 iterations with a batch size of 64, a learning rate of 0.6 and β equal to 5.5, and we finetuned the model for 200 epochs with the initial learning rate of 0.02 scheduled by cosine annealing scheduler and with a batch size of 256. For ImageNet, we trained the bottlenecks for 3000 iterations with a batch size of 64, a learning rate of 0.4 and β equal to 13, and we finetuned the model for 200 epochs with a batch size of 512 and with the initial learning rate of 0.006 scheduled by cosine annealing scheduler. Bottlenecks are optimized via Adam optimizer. All networks are retrained via the Stochastic Gradient Descent (SGD) optimizer, with momentum of 0.9 and

Table 1: Pruning results of five network architectures on CIFAR-10, sorted by FLOPs in descending order. Scores in brackets denote the pruning ratio in the compressed models. Unless specified otherwise, the accuracy before finetuning was re-computed by us using the code from the corresponding paper.

Method	Automatic	Top1-acc before finetuning	Top1-acc	↑↓	FLOPs (Pruning Ratio)	Params (Pruning Ratio)
VGG-16 (Simonyan and Zisserman 2015)		–	93.96%	0.0%	314.29M (0.0%)	14.99M (0.0%)
L1 (Li et al. 2017)		88.70%**	93.40%	-0.56%	206.00M (34.5%)	5.40M (64.0%)
CC-0.5 (Li et al. 2021)	✓	–	94.15%	+0.19%	154.00M (51.0%)	5.02M (66.5%)
AutoBot (Ours)	✓	**88.29%**	**94.19%**	**+0.23%**	**145.61M (53.7%)**	**7.53M (49.8%)**
CC-0.6 (Li et al. 2021)	✓	–	94.09%	+0.13%	123.00M (60.9%)	5.02M (73.2%)
HRank-65 (Lin et al. 2020a)		10.06%	92.34%	-1.62%	108.61M (65.4%)	2.64M (82.4%)
AutoBot (Ours)	✓	**82.73%**	**94.01%**	**+0.05%**	**108.71M (65.4%)**	**6.44M (57.0%)**
ITPruner (Zheng et al. 2021)	✓	10.00%*	94.00%	+0.04%	98.80 (68.6%)	–
ABCPruner (Lin et al. 2020b)	✓	10.00%*	93.08%	-0.88%	82.81M (73.7%)	1.67M (88.9%)
DCFF (Lin et al. 2021)		–	93.49%	-0.47%	72.77M (76.8%)	1.06M (92.9%)
AutoBot (Ours)	✓	**71.24%**	**93.62%**	**-0.34%**	**72.60M (76.9%)**	**5.51M (63.24%)**
VIBNet (Dai et al. 2018)	✓	–	91.50%	-2.46%	70.63M (77.5%)	– (94.7%)
ResNet-56 (He et al. 2016)		–	93.27%	0.0%	126.55M (0.0%)	0.85M (0.0%)
L1 (Li et al. 2017)		–	93.06%	-0.21%	90.90M (28.2%)	0.73M (14.1%)
HRank-50 (Lin et al. 2020a)		10.73%	93.17%	-0.10%	62.72M (50.4%)	0.49M (42.4%)
SCP (Kang and Han 2020)		–	93.23%	-0.04%	61.89M (51.1%)	0.44M (48.2%)
CC (Li et al. 2021)	✓	26.54%	93.64%	+0.37%	60.00M (52.6%)	0.44M (48.2%)
ITPruner (Zheng et al. 2021)	✓	10.00%*	93.43%	+0.16%	59.50 (53.0%)	–
FPGM (He et al. 2019)		17.44%	93.26%	-0.01%	59.40M (53.0%)	–
LFPC (He et al. 2020)		–	93.24%	-0.03%	59.10M (53.3%)	–
ABCPruner (Lin et al. 2020b)	✓	10.00%*	93.23%	-0.04%	58.54M (53.7%)	0.39M (54.1%)
DCFF (Lin et al. 2021)		–	93.26%	-0.01%	55.84M (55.9%)	0.38M (55.3%)
AutoBot (Ours)	✓	**85.58%**	**93.76%**	**+0.49%**	**55.82M (55.9%)**	**0.46M (45.9%)**
SCOP (Tang et al. 2020)		57.34%	93.64%	+0.37%	– (56.0%)	– (56.3%)
ResNet-110 (He et al. 2016)		–	93.5%	0.0%	254.98M (0.0%)	1.73M (0.0%)
L1 (Li et al. 2017)		–	93.30%	-0.20%	155.00M (39.2%)	1.16M (32.9%)
FPGM (He et al. 2019)		11.79%	93.74%	+0.24%	121.00M (52.5%)	–
HRank-58 (Lin et al. 2020a)		10.00%	93.36%	-0.14%	105.70M (58.5%)	0.70M (59.5%)
LFPC (He et al. 2020)		–	93.07%	-0.43%	101.00M (60.3%)	–
ABCPruner (Lin et al. 2020b)	✓	10.00%*	93.58%	+0.08%	89.87M (64.8%)	0.56M (67.6%)
DCFF (Lin et al. 2021)		–	93.80%	+0.30%	85.30M (66.5%)	0.56M (67.6%)
AutoBot (Ours)	✓	**84.37%**	**94.15%**	**+0.65%**	**85.28M (66.6%)**	**0.70M (59.5%)**
GoogLeNet (Szegedy et al. 2015)		–	95.05%	0.0%	1.53B (0.0%)	6.17M (0.0%)
L1 (Li et al. 2017)		–	94.54%	-0.51%	1.02B (33.3%)	3.51M (43.1%)
Random		10.00%	94.54%	-0.51%	0.96B (37.3%)	3.58M (42.0%)
HRank-54 (Lin et al. 2020a)		10.00%	94.53%	-0.52%	0.69B (54.9%)	2.74M (55.6%)
CC (Li et al. 2021)	✓	–	94.88%	-0.17%	0.61M (60.1%)	2.26M (63.4%)
ABCPruner (Lin et al. 2020b)	✓	10.00%*	94.84%	-0.21%	0.51B (66.7%)	2.46M (60.1%)
DCFF (Lin et al. 2021)		–	94.92%	-0.13%	0.46B (69.9%)	2.08M (66.3%)
HRank-70 (Lin et al. 2020a)		10.00%	94.07%	-0.98%	0.45B (70.6%)	1.86M (69.9%)
AutoBot (Ours)	✓	**90.18%**	**95.23%**	**+0.16%**	**0.45B (70.6%)**	**1.66M (73.1%)**
DenseNet-40 (Huang et al. 2017)		–	94.81%	0.0%	287.71M (0.0%)	1.06M (0.0%)
GAL-0.01 (Lin et al. 2019)		–	94.29%	-0.52%	182.92M (36.4%)	0.67M (36.8%)
AutoBot (Ours)	✓	**87.85%**	**94.67%**	**-0.14%**	**167.64M (41.7%)**	**0.76M (28.3%)**
HRank-40 (Lin et al. 2020a)		25.58%	94.24%	-0.57%	167.41M (41.8%)	0.66M (37.7%)
Variational CNN (Zhao et al. 2019)		–	93.16%	-1.65%	156.00M (45.8%)	0.42M (60.4%)
AutoBot (Ours)	✓	**83.20%**	**94.41%**	**-0.4%**	**128.25M (55.4%)**	**0.62M (41.5%)**
GAL-0.05 (Lin et al. 2019)		–	93.53%	-1.28%	128.11M (55.5%)	0.45M (57.5%)

*this method train the pruned model from scratch, instead of finetuning
**according to (Kim et al. 2020)

decay factor of 2×10^{-3} for CIFAR-10 and with momentum of 0.99 and decay factor of 1×10^{-4} for ImageNet.

4.2 Evaluation Metrics

We first evaluate the accuracy of the models. We measure it after finetuning, as is common in DNN pruning literature. However, in contrast to other works, we also measure it right after the pruning step (before finetuning) to show that our method effectively preserves the important filters compared to other methods. In addition, we adopt the FLOPs and number of parameters to measure the computational efficiency and model size.

4.3 Automatic Pruning on CIFAR-10

To demonstrate the improvement of our method, we firstly conduct automatic pruning with some of the most popular convolutional neural networks, namely VGG-16, ResNet-56/110, GoogLeNet, and DenseNet-40. Tab. 1 indicates experimental results with these architectures on CIFAR-10 for various number of FLOPs.

VGG-16 We performed on VGG-16 architecture with three different pruning ratios. Tab. 1 demonstrates that AutoBot can efficiently preserve initial Top-1 accuracy before finetuning, even under the same FLOPs reduction (e.g. 82.73% (proposed method) vs. 10.00% from 65.4% (HRank), 68.6% (ITPruner), and 73.7% (ABCPruner) of FLOPs reduction),

Table 2: Pruning results on ResNet-50 with ImageNet, sorted by FLOPs. Scores in brackets of "FLOPs" and "Params" denote the pruning ratio of FLOPs and number of parameters in the compressed models. Accuracy before finetuning was re-computed by us using the code from the corresponding paper.

Method	Automatic	Top1-acc before finetuning	Top1-acc	↑↓	Top5-acc	FLOPs (Pruning Ratio)	Params (Pruning Ratio)
ResNet-50 (He et al. 2016)		–	76.13%	0.0%	92.87%	4.11B (0.0%)	25.56M (0.0%)
ThiNet-50 (Luo, Wu, and Lin 2017)		–	72.04%	-4.09%	90.67%	– (36.8%)	– (33.72%)
FPGM (He et al. 2019)		0.25%	75.59%	-0.59%	92.27%	2.55B (37.5%)	14.74M (42.3%)
ABCPruner (Lin et al. 2020b)	✓	0.10%*	74.84%	-1.29%	92.31%	2.45B (40.8%)	16.92M (33.8%)
SFP (He et al. 2018)		–	74.61%	-1.52%	92.06%	2.38B (41.8%)	–
HRank-74 (Lin et al. 2020a)		0.09%	74.98%	-1.15%	92.33%	2.30B (43.7%)	16.15M (36.8%)
Taylor (Molchanov et al. 2019)		–	74.50%	-1.63%	–	– (44.5%)	– (44.9%)
DCFF (Lin et al. 2021)		–	75.18%	-0.95%	92.56%	2.25B (45.3%)	15.16M (40.7%)
ITPruner (Zheng et al. 2021)	✓	0.10%*	75.75%	-0.38%	–	2.23B (45.7%)	–
AutoPruner (Luo and Wu 2020b)	✓	–	74.76%	-1.37%	92.15%	2.09B (48.7%)	–
RRBP (Zhou et al. 2019)		–	73.00%	-3.13%	91.00%	–	– (54.5%)
AutoBot (Ours)	✓	**47.51%**	**76.63%**	**+0.50%**	**92.95%**	**1.97B (52.0%)**	**16.73M (34.5%)**
ITPruner (Zheng et al. 2021)	✓	0.10%*	75.28%	-0.85%	–	1.94B (52.8%)	–
GDP-0.6 (Lin et al. 2018)	✓	–	71.19%	-4.94%	90.71%	1.88B (54.0%)	–
SCOP (Tang et al. 2020)		4.26%	75.26%	-0.87%	92.53%	1.85B (54.6%)	12.29M (51.9%)
GAL-0.5-joint (Lin et al. 2019)		–	71.80%	-4.33%	90.82%	1.84B (55.0%)	19.31M (24.5%)
ABCPruner (Lin et al. 2020b)	✓	0.10%*	73.52%	-2.61%	91.51%	1.79B (56.6%)	11.24M (56.0%)
GAL-1 (Lin et al. 2019)		–	69.88%	-6.25%	89.75%	1.58B (61.3%)	14.67M (42.6%)
LFPC (He et al. 2020)		–	74.18%	-1.95%	91.92%	1.60B (61.4%)	–
GDP-0.5 (Lin et al. 2018)	✓	–	69.58%	-6.55%	90.14%	1.57B (61.6%)	–
DCFF (Lin et al. 2021)		–	75.60%	-0.53%	92.55%	1.52B (63.0%)	11.05M (56.8%)
DCFF (Lin et al. 2021)		–	74.85%	-1.28%	92.41%	1.38B (66.7%)	11.81M (53.8%)
AutoBot (Ours)	✓	**14.71%**	**74.68%**	**-1.45%**	**92.20%**	**1.14B (72.3%)**	**9.93M (61.2%)**
CURL (Luo and Wu 2020a)	✓	0.10%	73.39%	-2.74%	91.46%	1.13B (72.5%)	6.67M (73.9%)
GAL-1-joint (Lin et al. 2019)		–	69.31%	-6.82%	89.12%	1.11B (73.0%)	10.21M (60.1%)
DCFF (Lin et al. 2021)		–	73.81%	-2.32%	91.59%	1.02B (75.1%)	6.56M (74.3%)

* this method train the pruned model from scratch, instead of finetuning

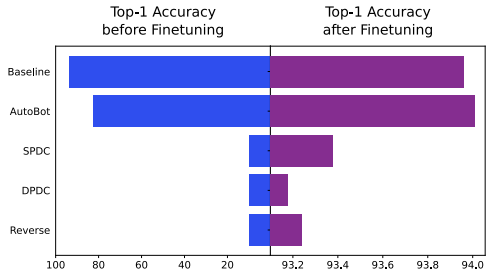

Figure 4: Top-1 accuracy before and after finetuning for different pruning strategies, on VGG-16. The strategies are detailed in Sec. 4.5

thus leading to a SOTA accuracy after finetuning. For instance, we get 71.24% and 93.62% accuracy before and after finetuning respectively when reducing the FLOPs by 76.9%. Our method even outperforms the baseline by 0.05% and 0.23% when reducing the FLOPs by 65.4% and 53.7%, respectively. As emphasized in Fig. 2, the per-layer filter pruning ratio is automatically determined by our method, according to the target FLOPs.

ResNet ResNet is an architecture characterized by its residual connections. Pruned model with our method can improve accuracy from 85.58% before finetuning to 93.76% after finetuning under a FLOPs reduction of 55.9% for ResNet-56, and from 84.37% before finetuning to 94.15% after finetuning under a FLOPs reduction of 66.6% for ResNet-110. Under similar or even smaller FLOPs, our approach accomplishes an excellent Top-1 accuracy compared to other existing magnitude-based or adaptive-based pruning methods and is beyond the baseline model's performance (93.27% for ResNet-56 and 93.50% for ResNet-110).

GoogLeNet GoogLeNet is a large architecture characterized by its parallel branches. Without any further processing, our initial accuracy of 90.18% after pruning under a FLOPs reduction of 70.6% (against 10% for HRank and ABCPruner for the similar compression ratio) leads to the SOTA accuracy of 95.23% after finetuning, outperforming recent papers such as DCFF and CC. Moreover, we also achieve a significant improvement in term of parameters reduction (73.1%), although it is not the primary focus of our method.

DenseNet-40 As ResNet, DenseNet-40 is an architecture based on residual connections. We experimented with two different target FLOPs, as shown in Tab. 1. Notably, we got an accuracy of 83.2% before finetuning and 94.41% after finetuning under a FLOPs reduction of 55.4%.

4.4 Automatic Pruning on ImageNet

To show the performance of our method on ILSVRC-2012, we chose the ResNet-50 architecture, made of 53 convolution layers followed by a fully-connected layer. Due to the complexity of this dataset (1,000 classes and millions of images), this task is more challenging than the compression of models on CIFAR-10. While existing pruning methods requiring to manually define the pruning ratio for each layer achieve reasonable performance, our global pruning method

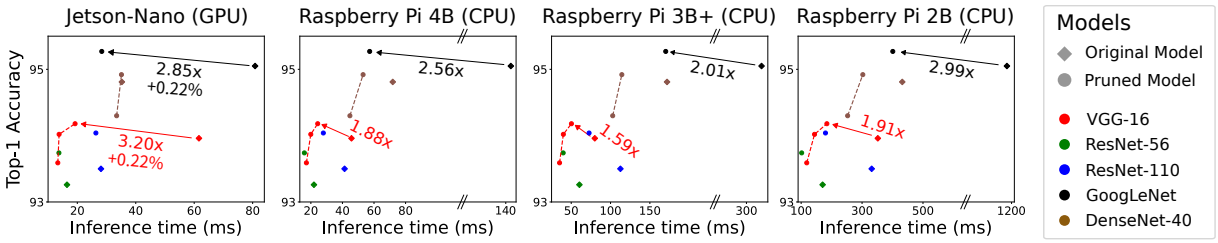

Figure 5: Performance comparison between original and pruned models in terms of accuracy (x-axis) and inference time (ms) (y-axis) using five different networks on CIFAR-10. Top-left is better performance.

allows competitive results in all evaluation metrics including Top-1 and Top-5 accuracy, FLOPs reduction as well as number of parameters reduction, as reported in Tab. 2. Under the high FLOPs compression of 72.3%, we obtain an accuracy of 74.68%, outperforming recent works including GAL (69.31%) and CURL (73.39%) with a similar compression. And under the compression of 52%, our method even outperforms the baseline by 0.5% and leaves all the previous methods behind by at least 1% by doing so. Therefore, the proposed method also works well on a complex dataset.

4.5 Ablation Study

Impact of Preserving the Accuracy To highlight the impact of preserving the accuracy during the pruning process, we compare the accuracy before and after finetuning of AutoBot with different pruning strategies in Fig. 4. To show the superiority of an architecture found by preserving the accuracy compared to a manual design, a comparison study is conducted by manually designing three different strategies: 1) Same Pruning, Different Channels (SPDC), 2) Different Pruning, Different Channels (DPDC), and 3) Reverse.

DPDC has the same FLOPs as the architecture found by AutoBot but uses a different per-layer pruning ratio proposed by Lin *et al*. (Lin et al. 2020a). To show the impact of a bad initial accuracy for finetuning, we propose the SPDC strategy that has the same per-layer pruning ratio as the architecture found by AutoBot but with randomly selected filters. We also propose to reverse the order of importance of the filters selected by AutoBot such that only the less important filters are pruned. By doing so, we can better appreciate the importance of the scores returned by AutoBot. In Fig. 4, we define this strategy as Reverse. This strategy gives a different per-layer pruning ratio than the architecture found by AutoBot. We evaluate the three strategies on VGG-16 with a pruning ratio of 65.4%, and we use the same finetuning conditions for all of them. We select the best accuracy among 3 runs. As shown in Fig. 4, these three different strategies give an initial accuracy of 10%. While the DPDC strategy gives an accuracy of 93.18% after finetuning, the SPDC strategy displays 93.38% accuracy, thus showing that an architecture found by preserving the initial accuracy gives better performance. Meanwhile, the Reverse strategy obtains 93.24%, which is surprisingly better than the hand-made architecture but, as expected, it underperforms the architecture found by AutoBot, even if we apply the SPDC strategy.

Deployment Test To highlight the improvement in real situations, we compare the inference speed-up of our compressed networks deployed on GPU-based (NVIDIA Jetson Nano) and CPU-based (Raspberry Pi 4, Raspberry Pi 3, and Raspberry Pi 2) edge devices. Specifications of these devices are available in the Supplementary Tab.2. The pruned models are converted into ONNX format. Fig. 5 shows the comparison study for inference times between the original pretrained models and our compressed models. We can show that inference time for our pruned models is improved in every target edge device (e.g. GoogleNet is 2.85× faster on Jetson-Nano and 2.56× faster on Raspberry Pi 4B with 0.22% increased accuracy). Especially, the speed is significantly better on GPU-based devices for single sequence of layers models (e.g. VGG-16 and GoogLeNet) whereas it improved the most on CPU-based devices for models with skip connections. More detailed results are available in the Supplementary Tab.3.

5 Limitations

While pruning with Autobot is a fast process, finding the hyper-parameters that most efficiently preserve the accuracy requires a hyper-parameter optimization step. However, our experiments highlight the relative stability of these hyper-parameters for different models on the same dataset. For instance, all our results on CIFAR10 presented in Tab. 1 were obtained with the same hyper-parameters.

For complex architectures, manually placing the bottlenecks can be challenging as it requires identifying which operations must be pruned together. It is interesting to notice that this could be solved with automation as these dependencies follow simple rules (e.g., in case of a skip connection, summed branches should be pruned together).

6 Conclusion

In this paper, we introduced AutoBot, a novel automatic pruning method focusing on FLOPs reduction. To determine which filters to prune, AutoBot employs trainable bottlenecks designed to preserve the channels that maximize the model accuracy while minimizing the FLOPs. Notably, these bottlenecks only require one epoch on 25.6% (CIFAR-10) or 15.0% (ILSVRC2012) of the dataset to be trained. Extensive experiments on various CNN architectures demonstrate that the proposed method is superior to previous channel pruning methods both before and after finetuning. Our paper is the first to compare accuracy before finetuning.

References

Alemi, A. A.; Fischer, I.; Dillon, J. V.; and Murphy, K. 2017. Deep Variational Information Bottleneck. In *International Conference on Learning Representations (ICLR)*.

Dai, B.; Zhu, C.; Guo, B.; and Wipf, D. P. 2018. Compressing Neural Networks using the Variational Information Bottleneck. In *International Conference on Machine Learning, (ICML)*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A Large-Scale Hierarchical Image Database. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.

Guo, Q.; Wang, X.; Wu, Y.; Yu, Z.; Liang, D.; Hu, X.; and Luo, P. 2020. Online Knowledge Distillation via Collaborative Learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

He, Y.; Ding, Y.; Liu, P.; Zhu, L.; Zhang, H.; and Yang, Y. 2020. Learning Filter Pruning Criteria for Deep Convolutional Neural Networks Acceleration. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

He, Y.; Kang, G.; Dong, X.; Fu, Y.; and Yang, Y. 2018. Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

He, Y.; Liu, P.; Wang, Z.; Hu, Z.; and Yang, Y. 2019. Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hu, H.; Peng, R.; Tai, Y.; and Tang, C. 2016. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. *arXiv:1607.03250*.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely Connected Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kang, M.; and Han, B. 2020. Operation-Aware Soft Channel Pruning using Differentiable Masks. In *International Conference on Machine Learning (ICML)*.

Kim, W.; Kim, S.; Park, M.; and Jeon, G. 2020. Neuron Merging: Compensating for Pruned Neurons. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning Multiple Layers of Features from Tiny Images. *Technical Report*.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2017. Pruning Filters for Efficient ConvNets. In *International Conference on Learning Representations (ICLR)*.

Li, Y.; Gu, S.; Mayer, C.; Gool, L. V.; and Timofte, R. 2020. Group Sparsity: The Hinge Between Filter Pruning and Decomposition for Network Compression. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Li, Y.; Lin, S.; Liu, J.; Ye, Q.; Wang, M.; Chao, F.; Yang, F.; Ma, J.; Tian, Q.; and Ji, R. 2021. Towards Compact CNNs via Collaborative Compression. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lin, M.; Ji, R.; Chen, B.; Chao, F.; Liu, J.; Zeng, W.; Tian, Y.; and Tian, Q. 2021. Training Compact CNNs for Image Classification using Dynamic-coded Filter Fusion. *arXiv:2107.06916*.

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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lin, M.; Ji, R.; Zhang, Y.; Zhang, B.; Wu, Y.; and Tian, Y. 2020b. Channel Pruning via Automatic Structure Search. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Lin, S.; Ji, R.; Li, Y.; Wu, Y.; Huang, F.; and Zhang, B. 2018. Accelerating Convolutional Networks via Global & Dynamic Filter Pruning. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Lin, S.; Ji, R.; Yan, C.; Zhang, B.; Cao, L.; Ye, Q.; Huang, F.; and Doermann, D. 2019. Towards Optimal Structured CNN Pruning via Generative Adversarial Learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, C.; and Wu, H. 2019. Channel Pruning based on Mean Gradient for Accelerating Convolutional Neural Networks. *Signal Processing*, 156: 84–91.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning Efficient Convolutional Networks through Network Slimming. In *IEEE International Conference on Computer Vision (ICCV)*.

Luo, J.; and Wu, J. 2020a. Neural Network Pruning With Residual-Connections and Limited-Data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Luo, J.; Wu, J.; and Lin, W. 2017. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. In *IEEE International Conference on Computer Vision (ICCV)*.

Luo, J.-H.; and Wu, J. 2020b. AutoPruner: An End-to-end Trainable Filter Pruning Method for Efficient Deep Model Inference. *Pattern Recognition*, 107: 107461.

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

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 *NIPS-W*.

Polino, A.; Pascanu, R.; and Alistarh, D. 2018. Model Compression via Distillation and Quantization. In *International Conference on Learning Representations (ICLR)*.

Qu, Z.; Zhou, Z.; Cheng, Y.; and Thiele, L. 2020. Adaptive Loss-Aware Quantization for Multi-Bit Networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Schulz, K.; Sixt, L.; Tombari, F.; and Landgraf, T. 2020. Restricting the Flow: Information Bottlenecks for Attribution. In *International Conference on Learning Representations (ICLR)*.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S. E.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going Deeper with Convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Tang, Y.; Wang, Y.; Xu, Y.; Tao, D.; Xu, C.; Xu, C.; and Xu, C. 2020. SCOP: Scientific Control for Reliable Neural Network Pruning. In *Advances in Neural Information Processing Systems (NIPS)*.

Tishby, N.; Pereira, F. C. N.; and Bialek, W. 2000. The Information Bottleneck Method. *arXiv:physics/0004057*.

Yang, Z.; Wang, Y.; Chen, X.; Guo, J.; Zhang, W.; Xu, C.; Xu, C.; Tao, D.; and Xu, C. 2021. HourNAS: Extremely Fast Neural Architecture Search Through an Hourglass Lens. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ye, J.; Lu, X.; Lin, Z.; and Wang, J. Z. 2018. Rethinking the Smaller-Norm-Less-Informative Assumption in Channel Pruning of Convolution Layers. In *International Conference on Learning Representations (ICLR)*.

Yeom, S.; Seegerer, P.; Lapuschkin, S.; Wiedemann, S.; Müller, K.; and Samek, W. 2021. Pruning by Explaining: A Novel Criterion for Deep Neural Network Pruning. *Pattern Recognition*, 115: 107899.

You, Z.; Yan, K.; Ye, J.; Ma, M.; and Wang, P. 2019. Gate Decorator: Global Filter Pruning Method for Accelerating Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Yu, R.; Li, A.; Chen, C.; Lai, J.; Morariu, V. I.; Han, X.; Gao, M.; Lin, C.; and Davis, L. S. 2018. NISP: Pruning Networks Using Neuron Importance Score Propagation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhao, C.; Ni, B.; Zhang, J.; Zhao, Q.; Zhang, W.; and Tian, Q. 2019. Variational Convolutional Neural Network Pruning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zheng, X.; Ma, Y.; Xi, T.; Zhang, G.; Ding, E.; Li, Y.; Chen, J.; Tian, Y.; and Ji, R. 2021. An Information Theory-inspired Strategy for Automatic Network Pruning. *arXiv:2108.08532*.

Zhou, Y.; Zhang, Y.; Wang, Y.; and Tian, Q. 2019. Accelerate CNN via Recursive Bayesian Pruning. In *IEEE/CVF International Conference on Computer Vision (ICCV)*.