Channel Pruning via Automatic Structure Search

Mingbao Lin[1], **Rongrong Ji**[*1,2], **Yuxin Zhang**[1], **Baochang Zhang**[3], **Yongjian Wu**[4], and **Yonghong Tian**[5]

[1]Media Analytics and Computing Laboratory, Department of Artificial Intelligence, School of Informatics, Xiamen University, China
[2]Peng Cheng Laboratory, Shenzhen, China
[3]School of Automation Science and Electrical Engineering, Beijing University, China
[4]Tencent Youtu Lab, Tencent Technology (Shanghai) Co., Ltd, China
[5]School of Electronics Engineering and Computer Science, Peking University, Beijing, China

Abstract

Channel pruning is among the predominant approaches to compress deep neural networks. To this end, most existing pruning methods focus on selecting channels (filters) by importance/optimization or regularization based on rule-of-thumb designs, which defects in sub-optimal pruning. In this paper, we propose a new channel pruning method based on artificial bee colony algorithm (ABC), dubbed as ABCPruner, which aims to efficiently find optimal pruned structure, *i.e.*, channel number in each layer, rather than selecting "important" channels as previous works did. To solve the intractably huge combinations of pruned structure for deep networks, we first propose to shrink the combinations where the preserved channels are limited to a specific space, thus the combinations of pruned structure can be significantly reduced. And then, we formulate the search of optimal pruned structure as an optimization problem and integrate the ABC algorithm to solve it in an automatic manner to lessen human interference. ABCPruner has been demonstrated to be more effective, which also enables the fine-tuning to be conducted efficiently in an end-to-end manner. Experiments on CIFAR-10 show that ABCPruner reduces 73.68% of FLOPs and 88.68% of parameters with even 0.06% accuracy improvement for VGGNet-16. On ILSVRC-2012, it achieves a reduction of 62.87% FLOPs and removes 60.01% of parameters with negligible accuracy cost for ResNet-152. The source codes can be available at https://github.com/lmbxmu/ABCPruner.

1 Introduction

The high demands in computing power and memory footprint of deep Convolutional Neural Networks (CNNs) prohibit their practical applications on edge computing devices such as smart phones or wearable gadgets. To address this problem, extensive studies have made on compressing CNNs. Prevalent techniques resort to quantization [Wang *et al.*, 2019a], decomposition [Zhang *et al.*, 2015], and pruning [Singh *et al.*, 2019]. Among them, channel pruning has been recognized as one of the most effective tools for compressing CNNs [Luo *et al.*, 2017; He *et al.*, 2017; He *et al.*, 2018b; He *et al.*, 2018a; Liu *et al.*, 2019a; Wang *et al.*, 2019b].

Channel pruning targets at removing the entire channel in each layer, which is straightforward but challenging because removing channels in one layer might drastically change the input of the next layer. Most cutting-edge practice implements channel pruning by selecting channels (filters) based on rule-of-thumb designs. Existing works follow two mainstreams. The first pursues to identify the most important filter weights in a pre-trained model, which are then inherited by the pruned model as an initialization for the follow-up fine-tuning [Hu *et al.*, 2016; Li *et al.*, 2017; He *et al.*, 2017; He *et al.*, 2018a]. It usually performs layer-wise pruning and fine-tuning, or layer-wise weight reconstruction followed by a data-driven and/or iterative optimization to recover model accuracy, both of which however are time-cost. The second typically performs channel pruning based on handcrafted rules to regularize the retraining of a full model followed by pruning and fine-tuning [Liu *et al.*, 2017; Huang and Wang, 2018; Lin *et al.*, 2019]. It requires human experts to design and decide hyper-parameters like sparsity factor and pruning rate, which is not automatic and thus less practical in compressing various CNN models. Besides, rule-of-thumb designs usually produce the sub-optimal pruning [He *et al.*, 2018b].

The motivation of our ABCPruner is two-fold. First, [Liu *et al.*, 2019b] showed that the essence of channel pruning lies in finding optimal pruned structure, *i.e.*, channel number in each layer, instead of selecting "important" channels. Second, [He *et al.*, 2018b] proved the feasibility of applying automatic methods for controlling hyper-parameters to channel pruning, which requires less human interference.

However, exhaustively searching for the optimal pruning structure is of a great challenge. Given a CNN with L layers, the combinations of pruned structure could be $\prod_{j=1}^{L} c_j$, where c_j is channel number in the j-th layer. The combina-

[*]Corresponding author: rrji@xmu.edu.cn

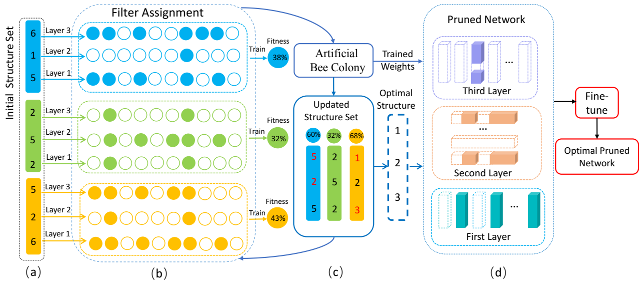


Figure 1: Framework of ABCPruner. (a) A structure set is initialized first, elements of which represent the preserved channel number. (b) The filters of the pre-trained model are randomly assigned to each structure. We train it for given epochs to measure its fitness. (c) Then, the ABC algorithm is introduced to update the structure set and the fitness is recalculated through (b). (b) and (c) will continue for some cycles. (d) The optimal pruned structure with best fitness is picked up, and the trained weights are reserved as a warm-up for fine-tuning the pruned network. (Best viewed with zooming in)

tion overhead is extremely intensive for deep CNNs[1], which is prohibitive for resource-limited scenarios.

In this paper, we introduce ABCPruner towards optimal channel pruning. To solve the above problem, two strategies are adopted in our ABCPruner. First, we propose to shrink the combinations by limiting the number of preserved channels to $\{0.1c_j, 0.2c_j, ..., \alpha c_j\}$ where the value of α falls in $\{10\%, 20\%, ..., 100\%\}$, which shows that there are 10α feasible solutions for each layer, and up to $\alpha\%$ percentage of channels in each layer will be preserved. This operation is interpretable since channel pruning usually preserves a certain percentage of channels. By doing so, the combinations will be significantly reduced to $(10\alpha)^L \ll \prod_{j=1}^{L} c_j$, making the search more efficient. Second, we obtain the optimal pruned structure through the artificial bee colony (ABC) algorithm [Karaboga, 2005]. ABC is automatic-based rather than hand-crafted, which thus lessens the human interference in determining the channel number. As shown in Fig. 1, we first initialize a structure set, each element of which represents the preserved channel number in each layer. The filter weights of the full model are randomly selected and assigned to initialize each structure. We train it for a given number of epochs to measure its fitness, *a.k.a*, accuracy performance in this paper. Then, ABC is introduced to update the structure set. Similarly, filter assignment, training and fitness calculation for the updated structures are conducted. We continue the search for some cycles. Finally, the one with the best fitness is considered as the optimal pruned structure, and its trained weights are reserved as a warm-up for fine-tuning. Thus, our ABCPruner can effectively implement channel pruning via automatically searching for the optimal pruned structure.

To our best knowledge, there exists only one prior work [Liu *et al.*, 2019a] which considers pruned structure in an automatic manner. Our ABCPruner differs from it in two aspects: First, the pruning in [Liu *et al.*, 2019a] is a two-stage scheme where a large PruningNet is trained in advance to predict weights for the pruned model, and then the evolutionary algorithm is applied to generate a new pruned structure. While our ABCPruner is one-stage without particularly designed network, which thus simplifies the pruning process. Second, the combinations in ABCPruner are drastically reduced to $(10\alpha)^L$ ($\alpha \in \{10\%, 20\%, ..., 100\%\}$), which provides more efficiency for the channel pruning.

Experimental results show ABCPruner can be well applied to most existing CNN models [Simonyan and Zisserman, 2015; Szegedy *et al.*, 2015; He *et al.*, 2016], which is better than both traditional pruning methods [Luo *et al.*, 2017; He *et al.*, 2017; Huang and Wang, 2018; Lin *et al.*, 2019] and automatic-based counterpart [Liu *et al.*, 2019a].

2 Related Work

Network Pruning. Network pruning can be categorized into either weight pruning or channel pruning. Weight pruning removes individual neurons in the filters or connections across different layers [Han *et al.*, 2015; Guo *et al.*, 2016; Li *et al.*, 2017; Aghasi *et al.*, 2017; Zhu and Gupta, 2017; Frankle and Carbin, 2019]. After pruning, a significant portion of CNNs' weights are zero, and thus the memory cost can be reduced by arranging the model in a sparse format. However, such a pruning strategy usually leads to an irregular network structure and memory access, which requires customized hardware and software to support practical speedup.

In contrast, channel pruning is especially advantageous by removing the entire redundant filters directly, which can be well supported by general-purpose hardware and BLAS libraries. Existing methods pursue pruning by channel selection based on rule-of-thumb designs. To that effect, many works aim to inherit channels based on the importance estimation of filter weights, *e.g.*, l_p-norm [Li *et al.*, 2017; He *et al.*, 2018a] and sparsity of activation outputs [Hu *et al.*, 2016]. [Luo *et al.*, 2017; He *et al.*, 2017] formulated channel pruning as an optimization problem, which selects the most representative filters to recover the accuracy of pruned network with minimal reconstruction error. However, it is still hard to implement in an end-to-end manner without iteratively pruning (optimization) and fine-tuning. Another group focuses on regularized-based pruning. For example, [Liu *et al.*, 2017] imposed sparsity constraint on the scaling factor of batch normalization layer, while [Huang and Wang, 2018; Lin *et al.*, 2019] proposed a sparsity-regularized mask for channel pruning, which is optimized through a data-driven selection or generative adversarial learning. However, these methods usually require another round of retraining and manual hyper-parameter analysis. Our work differs from traditional methods in that it is automatic and the corresponding fine-tuning is end-to-end, which has been demonstrated to be feasible by the recent work in [He *et al.*, 2018b].

AutoML. Traditional pruning methods involve human expert in hyper-parameter analysis, which hinders their practical applications. Thus, automatic pruning has attracted increasing attention, which can be regarded as a specific AutoML task. Most prior AutoML based pruning methods are implemented in a bottom-up and layer-by-layer man-

[1] For example, the combinations for VGGNet-16 are 2^{104}, 2^{448} for GoogLeNet, and 2^{1182} for ResNet-152.

ner, which typically achieve the automations through Q-value [Lin *et al.*, 2017], reinforcement learning [He *et al.*, 2018b] or an automatic feedback loop [Yang *et al.*, 2018]. Other lines include, but not limited to, constraint-aware optimization via an annealing strategy [Chen *et al.*, 2018] and sparsity-constraint regularization via a joint training manner [Luo and Wu, 2018]. Different from most prior AutoML pruning, our work is inspired by the recent work in [Liu *et al.*, 2019b] which reveals that the key of channel pruning lies in the pruned structure, *i.e.*, channel number in each layer, instead of selecting "important" channels.

3 The Proposed ABCPruner

Given a CNN model $\mathcal{N}$ that contains L convolutional layers and its filters set $\mathbf{W}$, we refer to $C = (c_1, c_2, ..., c_L)$ as the network structure of $\mathcal{N}$, where c_j is the channel number of the j-th layer. Channel pruning aims to remove a portion of filters in $\mathbf{W}$ while keeping a comparable or even better accuracy.

To that effect, traditional methods focus on selecting channels (filters) based on rule-of-thumb designs, which are usually sub-optimal [He *et al.*, 2018b]. Instead, our ABCPruner differs from these methods by finding the optimal pruned structure, *i.e.*, the channel number in each layer, and then solves the pruning in an automatic manner by integrating off-the-shelf ABC algorithm [Karaboga, 2005]. We illustrate our ABCPruner in Fig. 1.

3.1 Optimal Pruned Structure

Our channel pruning is inspired by the recent study in [Liu *et al.*, 2019b] which reveals that the key step in channel pruning lies in finding the optimal pruned structure, *i.e.*, channel number in each layer, rather than selecting "important" channels.

For any pruned model $\mathcal{N}'$, we denote its structure as $C' = (c'_1, c'_2, ..., c'_L)$, where $c'_j \leq c_j$ is the channel number of the pruned model in the j-th layer. Given the training set $\mathcal{T}_{train}$ and test set $\mathcal{T}_{test}$, we aim to find the optimal combination of C', such that the pruned model $\mathcal{N}'$ trained/find-tuned on $\mathcal{T}_{train}$ obtains the best accuracy. To that effect, we formulate our channel pruning problem as

$$(C')^* = \mathop{\arg\max}_{C'} \; acc\big(\mathcal{N}'(C', \mathbf{W}'; \mathcal{T}_{train}); \mathcal{T}_{test}\big), \qquad (1)$$

where $\mathbf{W}'$ is the weights of pruned model trained/fine-tuned on $\mathcal{T}_{train}$, and $acc(\cdot)$ denotes the accuracy on $\mathcal{T}_{test}$ for $\mathcal{N}'$ with structure C'. As seen, our implementation is one-stage where the pruned weights are updated directly on the pruned model, which differs from [Liu *et al.*, 2019a] where an extra large PruningNet has to be trained to predict weights for $\mathcal{N}'$, resulting in high complexity in the channel pruning.

However, the optimization of Eq. 1 is almost intractable. Detailedly, the potential combinations of network structure C' are extremely large. Exhaustively searching for Eq. 1 is infeasible. To solve this problem, we further propose to shrink the combinations in Sec. 3.2.

3.2 Combination Shrinkage

Given a network $\mathcal{N}$, the combination of pruned structure could be $\prod_{i=1}^{L} c_i$, which are extremely large and computationally prohibitive for the resource-limited devices. Hence, we further propose to constrain Eq. 1 as:

$$\begin{aligned} (C')^* = &\mathop{\arg\max}_{C'} \; acc\big(\mathcal{N}'(C', \mathbf{W}'; \mathcal{T}_{train}); \mathcal{T}_{test}\big), \\ &s.t. \;\; c'_i \in \{0.1c_i, 0.2c_i, ..., \alpha c_i\}^L, \end{aligned} \qquad (2)$$

where $\alpha = 10\%, 20\%, ..., 100\%$ is a pre-given constant and its influence is analyzed in Sec. 4.5. It denotes that for the i-th layer, at most α percentage of the channels in the pre-trained network $\mathcal{N}$ are preserved in $\mathcal{N}'$, *i.e.*, $c'_i \leq \alpha c_i$, and the value of c'_i is limited in $\{0.1c_i, 0.2c_i, ..., \alpha_i c_i\}$ [2]. Note that α is shared across all layers, which greatly relieves the burden of human interference to involve in the parameter analysis.

Our motivations of introducing α are two-fold: First, typically, a certain percentage of filters in each layer are preserved in the channel pruning, and α can play as the upbound of preserved filters. Second, the introduced α significantly decreases the combinations to $(10\alpha)^L$ $(\alpha \leq 1)$, which makes solving Eq. 1 more feasible and efficient.

Though the introduction of α significantly reduces the combinations of the pruned structure, it still suffers large computational cost to enumerate the remaining $(10\alpha)^L$ combinations for deep neural networks. One possible solution is to rely on empirical settings, which however requires manual interference. To solve the problem, we further automate the structure search in Sec. 3.3.

3.3 Automatic Structure Search

Instead of resorting to manual structure design or enumeration of all potential structure (channel number in each layer), we propose to automatically search the optimal structure.

In particular, we initialize a set of n pruned structures $\{C'_j\}_{j=1}^{n}$ with the i-th element c'_{ji} of C'_j randomly sampled from $\{0.1c_i, 0.2c_i, ..., \alpha c_i\}$. Accordingly, we obtain a set of pruned model $\{\mathcal{N}'_j\}_{j=1}^{n}$ and a set of pruned weights $\{\mathbf{W}'_j\}_{j=1}^{n}$. Each pruned structure C'_j represents a potential solution to the optimization problem of Eq. 2. Our goal is to progressively modify the structure set and finally pick up the best pruned structure. To that effect, we show that our optimal structure search for channel pruning can be optimized automatically by integrating off-the-shelf ABC algorithm [Karaboga, 2005]. We first outline our algorithm in Alg. 1. More details are elaborated below, which mainly contains three steps.

Employed Bee (Line 3 – Line 13). The employed bee generates a new structure candidate G'_j for each pruned structure C'_j. The i-th element of G'_j is defined as below:

$$g'_{ji} = \lceil c'_{ji} + r \cdot (c'_{ji} - c'_{gi}) \rfloor, \qquad (3)$$

where r is a random number within $[-1, +1]$ and $g \neq j$ denotes the g-th pruned structure. $\lceil \cdot \rfloor$ returns the value closest to the input in $\{0.1c_i, 0.2c_i, ..., \alpha c_i\}$.

Then, employed bee will decide if the generated candidate G'_j should replace C'_j according to their fitness, which is defined in our implementation as below:

$$fit_{C'_j} = acc\big(\mathcal{N}'_j(C'_j, \mathbf{W}'_j; \mathcal{T}_{train}); \mathcal{T}_{test}\big). \qquad (4)$$

[2] When $\alpha c_i < 1$, we set $c'_i = 1$ to preserve one channel.

Algorithm 1: ABCPruner

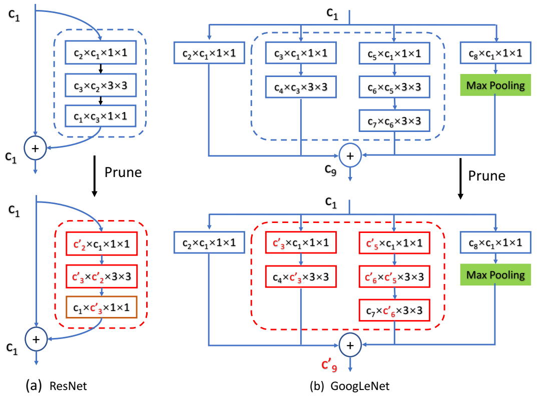
```
   Input: Cycles: $\mathcal{T}$, Upbound: $\alpha$, Number of pruned
          structures: $n$, Counter: $\{t_i\}_{i=1}^{n}$ = 0, Max times:
          $\mathcal{M}$.
   Output: Optimal pruned structure $(C')^*$.
1  Initialize the pruned structure set $\{C'_j\}_{j=1}^{n}$;
2  for $h = 1 \to \mathcal{T}$ do
3      for $j = 1 \to n$ do
4          generate candidate $G'_j$ via Eq. 3;
5          calculate the fitness of $C'_j$ and $G'_j$ via Eq. 4;
6          if $fit_{C'_j} < fit_{G'_j}$ then
7              $C'_j = G'_j$;
8              $fit_{C'_j} = fit_{G'_j}$;
9              $t_i = 0$;
10         else
11             $t_i$ = $t_i$ + 1;
12         end
13     end
14     for $j = 1 \to n$ do
15         calculate the probability of $P_j$ via Eq. 5;
16         generate a random number $\epsilon_j \in [0, 1]$;
17         if $\epsilon_j <= P_j$ then
18             generate candidate $G'_j$ via Eq. 3;
19             calculate the fitness of $G'_j$ via Eq. 4;
20             if $fit_{C'_j} < fit_{G'_j}$ then
21                 $C'_j = G'_j$;
22                 $fit_{C'_j} = fit_{G'_j}$;
23                 $t_i = 0$;
24             else
25                 $t_i$ = $t_i$ + 1;
26             end
27         end
28     end
29     for $j = 1 \to n$ do
30         if $t_i > \mathcal{M}$ then
31             re-initialize $C'_j$;
32         end
33     end
34 end
35 $(C')^* = \arg\max_{C'_j} \ acc\big(\mathcal{N}'_j(C'_j, \mathbf{W}'_j; \mathcal{T}_{train}); \mathcal{L}_{test}\big)$.
```

If the fitness of G'_j is better than C'_j, C'_j will be updated as $C'_j = G'_j$. Otherwise, C'_j will keep unchanged.

Onlooker Bee (Line 14 – Line 28). The onlooker bee further updates $\{C'_j\}_{j=1}^{n}$ following the rule of Eq. 3. Differently, a pruned structure C'_j is chosen with a probability related to its fitness, defined as:

$$P_j = 0.9 \cdot \frac{fit_{C'_j}}{\max(fit_{C'_j})} + 0.1. \tag{5}$$

Therefore, the better fitness of C'_j is, the higher probability of C'_j would be selected, which then produces a new and better pruned structure. By this, ABCPruner will automatically result in the optimal pruned structure progressively.


c_1
$c_2 \times c_1 \times 1 \times 1$
$c_3 \times c_2 \times 3 \times 3$
$c_1 \times c_3 \times 1 \times 1$
+
c_1
Prune
c_1
$c'_2 \times c_1 \times 1 \times 1$
$c'_3 \times c'_2 \times 3 \times 3$
$c_1 \times c'_3 \times 1 \times 1$
+
c_1
c_1
$c_2 \times c_1 \times 1 \times 1$
$c_3 \times c_1 \times 1 \times 1$
$c_5 \times c_1 \times 1 \times 1$
$c_8 \times c_1 \times 1 \times 1$
$c_4 \times c_3 \times 3 \times 3$
$c_6 \times c_5 \times 3 \times 3$
Max Pooling
$c_7 \times c_6 \times 3 \times 3$
+
c_9
Prune
c_1
$c_2 \times c_1 \times 1 \times 1$
$c'_3 \times c_1 \times 1 \times 1$
$c'_5 \times c_1 \times 1 \times 1$
$c_8 \times c_1 \times 1 \times 1$
$c_4 \times c'_3 \times 3 \times 3$
$c'_6 \times c'_5 \times 3 \times 3$
Max Pooling
$c_7 \times c'_6 \times 3 \times 3$
+
c'_9

(a) ResNet

(b) GoogLeNet

Figure 2: For ResNet, we prune the first two convolutional layers in each block and the residual mapping is not pruned. For GoogLeNet, the branches with only 1 × 1 convolutions are not pruned. For all pruned filters, the channels in the next layer are correspondingly removed. The c_j and c'_j denotes the channel (filter) number of pre-trained model and pruned model. (Best viewed with zooming in)

Scout Bee (Line 29 – Line 33). If a pruned structure C'_j has not been updated more than $\mathcal{M}$ times, the scout bee will re-initialize it to further produce a new pruned structure.

However, to calculate the fitness in Eq. 4, it has to train $\mathcal{N}'_j$ on $\mathcal{T}_{train}$ which is time-consuming and infeasible, especially when $\mathcal{T}_{train}$ is large-scale. To solve this problem, as shown in Fig. 1, given a potential combination of C', we first randomly pick up c'_j filters from the pre-trained model, which then serve as the initialization for the j-th layer of the pruned model $\mathcal{N}'$. Then, we train $\mathcal{N}'$ for some small epochs to obtain its fitness. Lastly, we give more epochs to fine-tune the pruned model with the best structure $(C')^*$, *a.k.a.*, best fitness. More details about the fine-tuning are discussed in Sec. 4.1.

4 Experiments

We conduct compression for representative networks, including VGGNet, GoogLeNet and ResNet-56/110 on CIFAR-10 [Krizhevsky *et al.*, 2009], and ResNet-18/34/50/101/152 on ILSVRC-2012 [Russakovsky *et al.*, 2015]. Fig. 2 outlines our pruning strategy for ResNet and GoogLeNet.

4.1 Implementation Details

Training Strategy. We use the Stochastic Gradient Descent algorithm (SGD) for fine-tuning with momentum 0.9 and the batch size is set to 256. On CIFAR-10, the weight decay is set to 5e-3 and we fine-tune the network for 150 epochs with a learning rate of 0.01, which is then divided by 10 every 50 training epochs. On ILSVRC-2012, the weight decay is set to 1e-4 and 90 epochs are given for fine-tuning. The learning rate is set as 0.1, and divided by 10 every 30 epochs.

Performance Metric. Channel number, FLOPs (floating-point operations), and parameters are used to measure the network compression. Besides, for CIFAR-10, top-1 accuracy of

Model	Top1-acc	↑↓	Channel	Pruned	FLOPs	Pruned	Parameters	Pruned
VGGNet-16 Base	93.02%	0.00%	4224	0.00%	314.59M	0.00%	14.73M	0.00%
VGGNet-16 ABCPruner-80%	93.08%	0.06%↑	1639	61.20%	82.81M	73.68%	1.67M	88.68%
GoogLeNet Base	95.05%	0.00%	7904	0.00%	1534.55M	0.00%	6.17M	0.00%
GoogLeNet ABCPruner-30%	94.84%	0.21%↓	6150	22.19%	513.19M	66.56%	2.46M	60.14%
ResNet-56 Base	93.26%	0.00%	2032	0.00%	187.62M	0.00%	0.85M	0.00%
ResNet-56 ABCPruner-70%	93.23%	0.03%↓	1482	27.07%	58.54M	54.13%	0.39M	54.20%
ResNet-110 Base	93.50%	0.00%	4048	0.00%	257.09M	0.00%	1.73M	0.00%
ResNet-110 ABCPruner-60%	93.58%	0.08%↑	2701	33.28%	89.87M	65.04%	0.56M	67.41%

Table 1: Accuracy and pruning ratio on CIFAR-10. We count the pruned channels, parameters and FLOPs for VGGNet-16 [Simonyan and Zisserman, 2015], GoogLeNet [Szegedy *et al.*, 2015] and ResNets with different depths of 56 and 110 [He *et al.*, 2016]. ABCPruner-α here denotes that at most α percentage of channels are preserved in each layer.

Model	Top1-acc	↑↓	Top5-acc	↑↓	Channel	Pruned	FLOPs	Pruned	Parameters	Pruned
ResNet-18 Base	69.66%	0.00%	89.08%	0.00%	4800	0.00%	1824.52M	0.00%	11.69M	0.00%
ResNet-18 ABCPruner-70%	67.28%	2.38%↓	87.67%	1.41%↓	3894	18.88%	1005.71M	44.88%	6.60M	43.55%
ResNet-18 ABCPruner-100%	67.80%	1.86%↓	88.00%	1.08%↓	4220	12.08%	968.13M	46.94%	9.50M	18.72%
ResNet-34 Base	73.28%	0.00%	91.45%	0.00%	8512	0.00%	3679.23M	0.00%	21.90M	0.00%
ResNet-34 ABCPruner-50%	70.45%	2.83%↓	89.69%	1.76%↓	6376	25.09%	1509.76M	58.97%	10.52M	51.76%
ResNet-34 ABCPruner-90%	70.98%	2.30%↓	90.05%	1.40%↓	6655	21.82%	2170.77M	41.00%	10.12M	53.58%
ResNet-50 Base	76.01%	0.00%	92.96%	0.00%	26560	0.00%	4135.70M	0.00%	25.56M	0.00%
ResNet-50 ABCPruner-70%	73.52%	2.49%↓	91.51%	1.45%↓	22348	15.86%	1794.45M	56.61%	11.24M	56.01%
ResNet-50 ABCPruner-80%	73.86%	2.15%↓	91.69%	1.27%↓	22518	15.22%	1890.60M	54.29%	11.75M	54.02%
ResNet-101 Base	77.38%	0.00%	93.59%	0.00%	52672	0.00%	7868.40M	0.00%	44.55M	0.00%
ResNet-101 ABCPruner-50%	74.76%	2.62%↓	92.08%	1.51%↓	41316	21.56%	1975.61M	74.89%	12.94M	70.94%
ResNet-101 ABCPruner-80%	75.82%	1.56%↓	92.74%	0.85%↓	43168	17.19%	3164.91M	59.78%	17.72M	60.21%
ResNet-152 Base	78.31%	0.00%	93.99%	0.00%	75712	0.00%	11605.91M	0.00%	60.19M	0.00%
ResNet-152 ABCPruner-50%	76.00%	2.31%↓	92.90%	1.09%↓	58750	22.40%	2719.47M	76.57%	15.62M	74.06%
ResNet-152 ABCPruner-70%	77.12%	1.19%↓	93.48%	0.51%↓	62368	17.62%	4309.52M	62.87%	24.07M	60.01%

Table 2: Accuracy and pruning ratio on ILSVRC-2012. We count pruned channels, parameters and FLOPs for ResNets with different depths of 18, 34, 50, 101 and 152 [He *et al.*, 2016]. ABCPruner-α here denotes that at most α percentage of channels are preserved in each layer.

pruned models are provided. For ILSVRC-2012, both top-1 and top-5 accuracies are reported.

For each structure, we train the pruned model $\mathcal{N}'$ for two epochs to obtain its fitness. We empirically set $\mathcal{T}$=2, n=3, and $\mathcal{M}$=2 in the Alg. 1. Due to the page limit, we do not provide ablation studies for these hyper-parameters.

4.2 Results on CIFAR-10

We conduct our experiments on CIFAR-10 with three classic deep networks including VGGNet, GoogLeNet and ResNets. The experiment results are reported in Tab. 1.

VGGNet. For VGGNet, the 16-layer (13-Conv + 3FC) model is adopted for compression on CIFAR-10. We remove 61.20% channels, 73.68% FLOPs and 88.68% parameters while still keeping the accuracy at 93.08%, which is even slightly better than the baseline model. This greatly facilitates VGGNet model, a popular backbone for object detection and semantic segmentation, to be deployed on mobile devices.

GoogLeNet. For GoogLeNet, as can be seen from Tab. 1, we remove 22.19% inefficient channels with only 0.21% performance drop of accuracy. Besides, nearly 66.56% FLOPs computation is saved and 60.14% parameters are reduced. Therefore, ABCPruner can be well applied even on the compact-designed GoogLeNet.

ResNets. For ResNets, we choose two different depths, including ResNet-56 and ResNet-110. As seen from Tab. 1, the pruning rates with regard to channels, FLOPs and parameters increase from ResNet-56 to ResNet-110. Detailedly, the channels reduction raises from 27.07% to 33.28%; the FLOPs reduction raises from 54.13% to 65.05%; and the parameters reduction raises from 54.20% to 67.41%. To explain, ResNet-110 is deeper and more over-parameterized. ABCPruner can automatically find out the redundancies and remove them. Besides, it retains comparable accuracies against the baseline model, which verifies the efficacy of ABCPruner in compressing the residual-designed networks.

4.3 Results on ILSVRC-2012

We further perform our method on the large-scale ILSVRC-2012 for ResNets with different depths, including

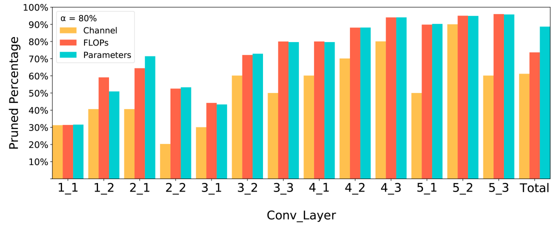


Figure 3: The pruned percentage of each layer for VGGNet on CIFAR-10 when $\alpha = 80\%$.

18/34/50/101/152. We present two different pruning rates for each network and the results are reported in Tab. 2.

From Tab. 2, we have two observations. First observation is that the performance drops on ILSVRC-2012 are more than these on CIFAR-10. The explanations are two-fold: On one hand, the ResNet itself is a compact-designed network, there might exist less redundant parameters. On the other hand, ILSVRC-2012 is a large-scale dataset and contains 1,000 categories, which is much complex than the small-scale CIFAR-10 with only 10 categories. Second observation comes that ABCPruner obtains higher pruning rates and less accuracy drops as the depth of network increases. To explain, compared with shallow ResNets, *e.g.*, ResNet-18/34, the deeper ResNets, *e.g.*, ResNet-50/191/152 contain relatively more redundancies, which means more pointless parameters are automatically found and removed by ABCPruner.

In-depth Analysis. Combining Tab. 1 and Tab. 2, we can see that ABCPruner can well compress popular CNNs while keeping a better or at least comparable accuracy performance against the full model. Its success lies in the automatically optimal pruned structure search. For deeper analysis, we display the layer-wise pruning results by ABCPruner-80% for VGGNet-16 in Fig. 3. ABCPruner-80% denotes that at most 80% percentage of the channels are preserved in each layer. As can be seen, over 20% channels are removed for most layers and the pruning rate differs across different layers. Hence, the ABCPruner can automatically obtain optimal pruned structure which then feeds back good performance.

Besides, we also observe that the channel pruning rates for GoogLeNet and ResNets are not so high as for VGGNet-16. The explanation can be found in Fig. 2. For GoogLeNet, we do not remove the 1×1 convolutional branches, and for ResNet, we do not remove the residual mapping, both of which contains abundant channels but less parameters. Hence, the pruning rate of channels is relatively less than VGGNet-16.

4.4 Comparison with Other Methods

In Tab. 3, we compare ABCPruner with traditional methods including [Luo *et al.*, 2017; He *et al.*, 2017; Huang and Wang, 2018; Lin *et al.*, 2019], and automatic method [Liu *et al.*, 2019a][3].

Effectiveness. The results in Tab. 3 show that ABCPruner obtains better FLOPs reduction and accuracy performance. Compared with importance-based pruning [Luo *et al.*, 2017; He *et al.*, 2017] and regularized-based pruning [Huang and Wang, 2018; Lin *et al.*, 2019], ABCPruner also advances in its end-to-end fine-tuning and automatic structure search. It effectively verifies the finding in [Liu *et al.*, 2019b] that optimal pruned structure is more important in channel pruning, rather than selecting "important" channels. In comparison with the automatic-based MetaPruning [Liu *et al.*, 2019a], the superiority of ABCPruner can be attributed to the one-stage design where the fitness of pruned structure is directly measured in the target network without any additionally introduced network as in [Liu *et al.*, 2019a]. Hence, ABCPruner is more advantageous in finding the optimal pruned structure.

Model	FLOPs	Top1-acc	Baseline-acc	Epochs
ThiNet-30	1.10G	68.42%	76.01%	244 (196 + 48)
ABCPruner-30%	0.94G	70.29%	76.01%	102 (12+90)
SSS-26	2.33G	71.82%	76.01%	100
GAL-0.5	2.33G	71.95%	76.01%	150 (90 + 60)
GAL-0.5-joint	1.84G	71.80%	76.01%	150 (90 + 60)
ThiNet-50	1.71G	71.01%	76.01%	244 (196 + 48)
ABCPruner-50%	1.30G	72.58%	76.01%	102 (12+90)
SSS-32	2.82G	74.18%	76.01%	100
CP	2.73G	72.30%	76.01%	206 (196 + 10)
ABCPruner-100%	2.56G	74.84%	76.01%	102 (12+90)
MetaPruning-0.50	1.03G	69.92%	76.01%	160 (32 + 128)
ABCPruner-30%	0.94G	70.29%	76.01%	102 (12+90)
MetaPruning-0.75	2.26G	72.17%	76.01%	160 (32 + 128)
ABCPruner-50%	1.30G	72.58%	76.01%	102 (12+90)
MetaPruning-0.85	2.92G	74.49%	76.01%	160 (32 + 128)
ABCPruner-100%	2.56G	74.84%	76.01%	102 (12+90)

Table 3: The FLOPs reduction, top-1 accuracy and training efficiency of ABCPruner and SOTAs including ThiNet [Luo *et al.*, 2017], CP [He *et al.*, 2017], SSS [Huang and Wang, 2018], GAL [Lin *et al.*, 2019], and MetaPruning [Liu *et al.*, 2019a] for ResNet-50 [He *et al.*, 2016] on ILSVRC-2012.

Efficiency. We also display the overall training epochs for all methods in Tab. 3. As seen, ABCPruner requires less training epochs of 102 compared with others. To analyze, 12 epochs are used to search for the optimal pruned structure and 90 epochs are adopted for fine-tuning the optimal pruned network. We note that the importance-based pruning methods [Luo *et al.*, 2017; He *et al.*, 2017] are extremely inefficient since they require layer-wise pruning or optimization (196 epochs). Besides, 48 epochs [Luo *et al.*, 2017] and 10 epochs [He *et al.*, 2017] are used to fine-tune the network. [Huang and Wang, 2018] consume similar training time with ABCPruner, but suffers more accuracy drops and less FLOPs reduction. [Lin *et al.*, 2019] cost 90 epochs for retraining the network and additional 60 epochs are required for fine-tuning. Besides, ABCPruner also shows its better efficiency against the automatic-based counterpart [Liu *et al.*, 2019a] which is a two-stage method where 32 epochs are first adopted to train the large PruningNet and 128 epochs are used for fine-tuning. Hence, ABCPruner is more efficient compared to the SOTAs.

4.5 The Influence of α

In this section, we take ResNet-56 on CIFAR-10 as an example and analyze the influence of the introduced constant α,

[3] We have carefully fixed the codes provided by the authors and re-run the experiments using our baseline of ResNet-50.

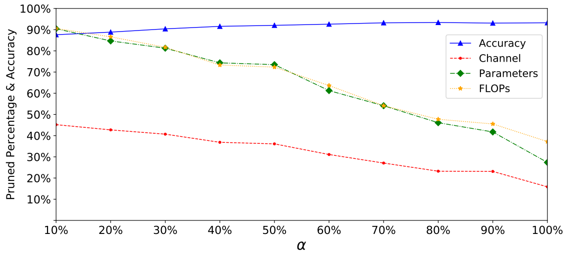


Figure 4: The influence of α for ResNet-56 on CIFAR-10. Generally, large α results in better accuracy but less pruning rate.

which represents the upbound of preserved channel percentage in each layer. It is intuitive that larger α leads less reductions of channel, parameters and FLOPs, but better accuracy performance. The experimental results in Fig. 4 demonstrate this assumption. To balance the accuracy performance and model complexity reduction, in this paper, we set $\alpha = 70\%$ as shown in Tab. 1.

5 Conclusion

In this paper, we introduce a novel channel pruning method, termed as ABCPruner. ABCPruner proposes to find the optimal pruned structure, *i.e.*, channel number in each layer via an automatic manner. We first propose to shrink the combinations of pruned structure, leading to efficient search of optimal pruned structure. Then, the artificial bee colony is integrated to achieve the optimal pruned structure search in an automatic manner. Extensive experiments on popular CNNs have demonstrated the efficacy of ABCPruner over traditional channel pruning methods and automatic-based counterpart.

References

[Aghasi *et al.*, 2017] Alireza Aghasi, Afshin Abdi, Nam Nguyen, and Justin Romberg. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *NeurIPS*, 2017.

[Chen *et al.*, 2018] Changan Chen, Frederick Tung, Naveen Vedula, and Greg Mori. Constraint-aware deep neural network compression. In *ECCV*, 2018.

[Frankle and Carbin, 2019] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[Guo *et al.*, 2016] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *NeurIPS*, 2016.

[Han *et al.*, 2015] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPs*, 2015.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[He *et al.*, 2017] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

[He *et al.*, 2018a] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*, 2018.

[He *et al.*, 2018b] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *CVPR*, 2018.

[Hu *et al.*, 2016] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[Huang and Wang, 2018] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *ECCV*, 2018.

[Karaboga, 2005] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report, 2005.

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

[Li *et al.*, 2017] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

[Lin *et al.*, 2017] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *NeurIPS*, 2017.

[Lin *et al.*, 2019] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *CVPR*, 2019.

[Liu *et al.*, 2017] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[Liu *et al.*, 2019a] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *ICCV*, 2019.

[Liu *et al.*, 2019b] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.

[Luo and Wu, 2018] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*, 2018.

[Luo *et al.*, 2017] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

[Russakovsky *et al.*, 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

[Simonyan and Zisserman, 2015] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[Singh *et al.*, 2019] Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay P. Namboodiri. Play and prune: Adaptive filter pruning for deep model compression. In *IJCAI*, 2019.

[Szegedy *et al.*, 2015] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[Wang *et al.*, 2019a] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *CVPR*, 2019.

[Wang *et al.*, 2019b] Wenxiao Wang, Cong Fu, Jishun Guo, Deng Cai, and Xiaofei He. Cop: Customized deep model compression via regularized correlation-based filter-level pruning. In *IJCAI*, 2019.

[Yang *et al.*, 2018] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*, 2018.

[Zhang *et al.*, 2015] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *CVPR*, 2015.

[Zhu and Gupta, 2017] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. In *ICLR*, 2017.