2022 26th International Conference on Pattern Recognition (ICPR)
August 21-25, 2022, Montréal, Québec, Canada

AACP: Model Compression by Accurate and Automatic Channel Pruning

Lanbo Lin, Shengjie Chen, Yujiu Yang and Zhenhua Guo
Tsinghua Shenzhen International Graduate School
Tsinghua University, Shenzhen, China
Email: {llb19, chen-sj17}@mails.tsinghua.edu.cn, {yang.yujiu, zhenhua.guo}@sz.tsinghua.edu.cn

2022 26th International Conference on Pattern Recognition (ICPR) | 978-1-6654-9062-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICPR56361.2022.9956562

***Abstract*—Channel pruning is formulated as a neural architecture search (NAS) problem recently, which achieves impressive performance in model compression. However, prior arts only considered one kind of constraint (FLOPs, inference latency or model size) when pruning a neural network. This will lead to an unbalanced-pruning problem, where the FLOPs of a pruned network are under budget but the inference latency and model size are still unaffordable. Another challenge is that the supernet training process of typical NAS-based channel pruning methods is computationally expensive. To address these problems, we propose a novel Accurate and Automatic Channel Pruning (AACP) method. Firstly, we impose multiple constraints on channel pruning to address the unbalanced-pruning problem. To solve this complicated multi-objective problem, AACP proposes Improved Differential Evolution (IDE) algorithm which is more effective than typical evolutionary algorithms in searching for optimal architectures. Secondly, AACP proposes a Pruned Structure Accuracy Estimator (PSAE) which can estimate the performance of sub-networks without training a supernet and speeds up the performance estimation process. Our method achieves state-of-the-art performance on several benchmarks. On CIFAR10, our method reduces 65% FLOPs of ResNet110 with an improvement of 0.26% top-1 accuracy. On ImageNet, we reduce 42% FLOPs of ResNet50 with a small loss of 0.06% top-1 accuracy. The code is available at https://github.com/linlb11/AACP.**

I. Introduction

Channel pruning is an effective way of model compression which aims to reduce FLOPs and redundant parameters of a CNN while keeping its high performance. Recently, Liu et al.[1] find that the pruned architecture is more important than inheriting "important" weights from an unpruned model. Inspired by this discovery, some works[2], [3], [4], [5] focus on searching the optimal architecture, i.e. channel numbers, from an unpruned model and achieve impressive performance. However, there are still some challenges in this field.

Firstly, prior works only considered one kind of constraint when pruning a CNN. Some works[6], [7], [8], [9] only try to reduce the model size, i.e., the total number of channels in a network. Some works[10], [2] only consider the floating-point operations (FLOPs) of a network. The others[11], [12] directly impose a constraint on inference latency.

The relationship among FLOPs, model size and latency is not linear, so a decline in one aspect does not mean a decline in other aspects. For example, as illustrated in Fig.1, if we try to keep less than 40% FLOPs when pruning a network, the pruned network may have a wide range of possible model size and inference latency. This may lead to a pruned model whose model size and inference latency are hardly reduced after pruning, which we called the **unbalanced-pruning** problem. One way to solve this problem is to impose multiple constraints on channel pruning. To the best of our knowledge, it is the first time to discuss the unbalanced-pruning problem and impose multiple constraints on channel pruning.

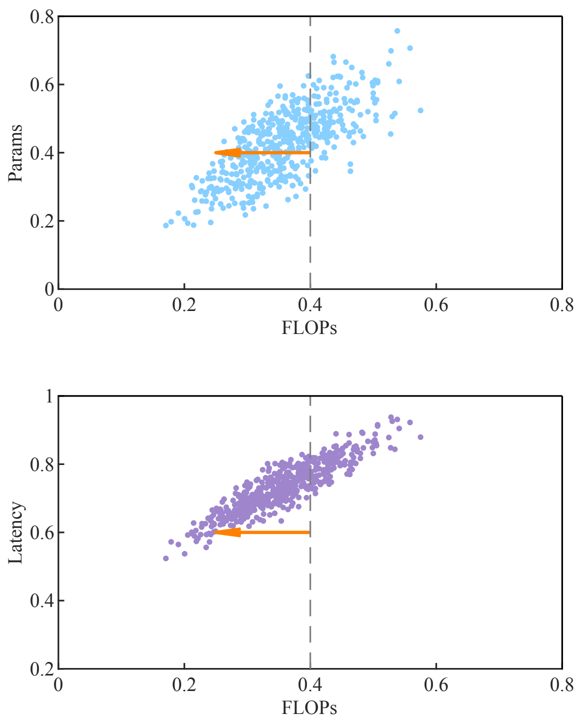

Fig. 1. The relationship between FLOPs and model size/inference latency. We randomly sample 500 sub-networks from ResNet50 and illustrate the distribution of FLOPs and model size/inference latency of those sub-networks. All values are normalized to $[0, 1]$. The arrows and dotted lines stand for keeping less than 40% FLOPs. Prior arts of channel pruning only consider one kind of constraint, typically FLOPs, which cannot control the model size and inference latency simultaneously and accurately as illustrated.

Secondly, prior NAS-based channel pruning methods are usually computationally expensive in the performance estimation process. Either training a one-shot supernet[5], [3], [2] or fine-tuning the sub-networks[4] requires huge computational cost. One approach to speed up the performance estimation process is to combine the efficient parts of these methods.

In this paper, we propose a novel Accurate and Automatic

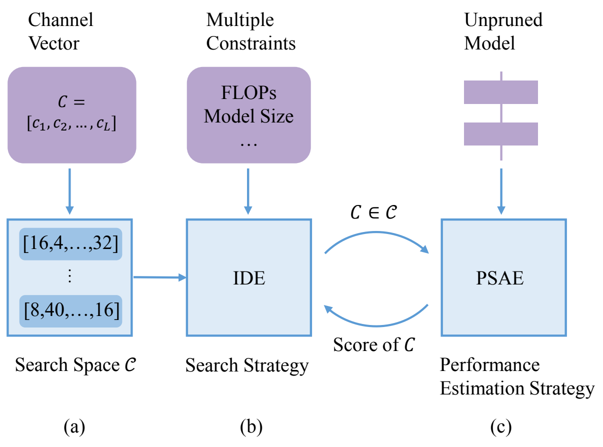

Fig. 2. The framework of AACP. (a) Search space. We represent the architecture of a CNN as a vector of its channel numbers. (b) Search strategy. AACP proposes IDE to search for optimal pruned structures under multiple constraints. (c) Performance Estimation Strategy. Given a structure vector C, AACP adopts PSAE to estimate its performance. A pre-trained unpruned network is applied to share weights with pruned models.

Channel Pruning (AACP) method to alleviate these problems, as illustrated in Fig.2. Firstly, we propose a novel Improved Differential Evolution (IDE) algorithm to efficiently search for optimal structures under multiple constraints. The reasons we adopt IDE are twofold: 1) Evolutionary algorithms (EAs) have achieved great performance in channel pruning and NAS. 2) Multiple constraints are more complex than a single constraint which requires a more powerful EA. Secondly, we propose a novel performance estimation strategy named Pruned Structure Accuracy Evaluator (PSAE) to speed up the process of performance estimation. In PSAE, we combine the efficient parts in [4], [3], [5] and use a one-shot unpruned model to estimate the performance of sub-networks, instead of training a one-shot supernet or fine-tune the sub-networks. Note that in this paper, a supernet refers to a particularly designed network, which shares weights with its sub-networks[3], [2], [5]. While an unpruned network refers to a standard network such as ResNet and MobileNet. Our main contributions can be summarized as:

- We analyze the unbalanced-pruning problem in channel pruning and propose a novel channel pruning method AACP to address this problem. Our method can perform accurate pruning under multiple constraints.
- We propose a novel IDE algorithm, which is efficient and effective not just in channel pruning, but also in general neural architecture search. We achieve state-of-the-art performance on several benchmarks.
- Our method largely speeds up the process of performance estimation because of the proposed PSAE, which is simpler than other NAS-based channel pruning methods.

II. Related Works

Neural architecture search. Channel pruning can be formulated as a neural architecture search problem, except that the search space is limited to channel numbers. Our work is most closely related to one-shot NAS[13], [14], [15]. One-shot NAS methods firstly train an over-parameterized supernet for efficient performance evaluation. Secondly, search strategies are applied to search for optimal architectures. Different from searching different types of operations in general NAS methods, we focus on searching structures with different channel numbers. Besides, we impose multiple constraints instead of a single constraint in the searching stage.

Channel Pruning. Neural network pruning can be summarized as weight pruning and channel pruning. We focus on channel pruning methods, which remove the whole filters, leading to structured sparsity. Some works utilize certain metrics to evaluate the importance of filters. Li et al.[6] propose to use l_1-norm to sort filters. He et al.[9] calculate the geometric median of the filters within the same layer and remove those filters that are closest to the geometric median. Another idea is to impose sparsity regularization on filters or channels by changing the loss function. Liu et al.[8] propose to impose sparsity constraints on the scaling factors of batch normalization layers. Tang et al.[16] impose l_1-norm penalty on the channel saliency and prunes a network dynamically according to input data.

Recently, the architecture of a network, i.e. channel numbers, is thought to be crucial in channel pruning, inspired by [1]. We follow this discovery to search for optimal architectures. He et al.[10] utilize reinforcement learning to search the channel numbers layer by layer. Yu et al.[3] and Liu et al.[5] firstly train a one-shot supernet and then search the efficient architectures within the one-shot model. Lin et al.[4] find an optimal pruned structure based on the Artificial Bee Colony algorithm. Guo et al.[2] make the channel pruning differentiable by modeling it as a Markov process. These methods have high computational costs either in the process of training a large one-shot supernet[3], [5], [2] or in the process of fine-tuning sub-networks[4]. Our method doesn't need to train a one-shot model or fine-tune pruned models in the searching stage, which largely speeds up the process of finding the optimal structures.

III. Methodology

A. *Definition of channel pruning*

Given an unpruned model $\mathcal{M}$ with L convolutional layers, we can represent the architecture of $\mathcal{M}$ as a vector $C = [c_1, c_2, ..., c_L]$, where c_i refers to the output channel number of the i-th convolutional layer. We denote a pruned model as $\mathcal{M}^{'}$ and the architecture of $\mathcal{M}^{'}$ as $C^{'} = [c_1^{'}, c_2^{'}, \cdots, c_L^{'}]$. Note that $0 < c_i^{'} \leq c_i, c_i^{'} \in \mathbb{Z}$. The goal of channel pruning is to find an optimal structure vector $C^{'*} = [c_1^{'*}, c_2^{'*}, ..., c_L^{'*}]$ under certain sparsity constraints.

We introduce $r_f \in [0, 1]$ as the pruning rate of FLOPs and $r_p \in [0, 1]$ as the pruning rate of parameters. Note that other kinds of constraints, such as inference latency, can be dealt with in the same way because evolutionary computation is a black-box optimization algorithm. Given a pruned model $\mathcal{M}^{'}$

with structure $C^{'}$ and an unpruned model $\mathcal{M}$ with structure C, we have:

$$r_f(C^{'}) = 1 - \frac{FLOPs(C^{'})}{FLOPs(C)}$$
$$r_p(C^{'}) = 1 - \frac{Params(C^{'})}{Params(C)} \tag{1}$$

where $FLOPs(*)$ is the calculation function of FLOPs and $Params(*)$ is the calculation function of parameters. We have $0 \leq r_f(*) \leq 1$ and $0 \leq r_p(*) \leq 1$. Therefore, our channel pruning problem can be formulated as:

$$\begin{aligned} C^{'*} &= \max_{C^{'}}(acc(C^{'};C)) \\ s.t. \quad & r_f(C^{'}) \geq R_f \\ & r_p(C^{'}) \geq R_p \end{aligned} \tag{2}$$

where $C^{'*}$ is the optimal structure vector, R_f is the target of FLOP pruning rate, R_p is the target of parameter pruning rate and $acc(*;C)$ is the validation accuracy of a model obtained by pruning C. Note that if we set $R_p = 0$ or $R_f = 0$, there is only one constraint in this problem, as with other channel pruning methods.

B. Improved Differential Evolution Algorithm

Considering the success of EAs in channel pruning[5], [4] and the complexity of multi-constraint optimization problem Eq.2, we make some improvements based on the standard DE[17] and then apply it to our problem. The common problem of existing EAs is that they are easy to get stuck in locally optimal solutions. We proposed a novel **re-initialization** step to help explore new areas and find the optimal solution. Our IDE algorithm is illustrated in Alg.1.

Population. The population X is consist of N individuals $X_n, n = 1, \ldots, N$. Each individual X_i is a structure vector. The population of generation t can be represented as:

$$X^t = \{X_1^t, X_2^t, ..., X_N^t\} \tag{3}$$

where N is the size of the population, X_n^t is the n-th individual of generation t.

Mutation. At iteration t, we randomly select three individuals X_p^t, X_q^t and X_r^t from X^t. Then a new candidate of individual V_n^{t+1} can be generated by:

$$V_n^{t+1} = X_p^t + F \times (X_q^t - X_r^t) \tag{4}$$

where $F \in [0, 2]$ is the differential weight. In our experiment, F is set to 0.5.

Crossover. While mutation introduces individual-level diversity to the population, crossover introduces gene-level diversity. Crossover produces candidate individual by:

$$U_{n,j}^{t+1} = \begin{cases} V_{n,j}^{t+1}, & if \quad rand < CR \\ X_{n,j}^t, & otherwise \end{cases} \tag{5}$$

where $V_n^{t+1} = [V_{n,1}^{t+1}, \ldots, V_{n,L}^{t+1}]$ is the candidate individual produced by mutation, $rand \in [0, 1]$ is a random number and

Algorithm 1 Improved Differential Evolution Algorithm

Input: Target of FLOP pruning rate R_f, target of parameter pruning rate R_p, iteration number T, population size N.

Output: The optimal pruned structure $C^{'*}$

```
1:  Initialize the Population $X^0 = \{X_n^0\}_{n=1,2,...,N}$;
2:  for each $t \in [1, T]$ do
3:      for each $n \in [1, N]$ do
4:          Perform mutation according to Eq.4;
5:          Perform crossover according to Eq.5;
6:          Perform selection according to Eq.6;
7:          Perform re-initialization according to Eq.7.
8:      end for
9:  end for
10: return $C^{'*} = \max_{n=1,...,N} fitness(X_n^T)$.
```

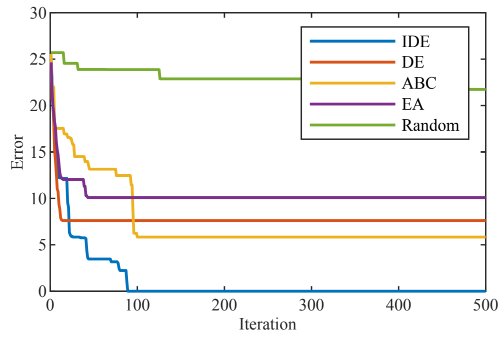

Fig. 3. Experimental results of Eq.8 by IDE (ours), DE[17], ABC[4], EA[5] and random search.

$CR \in [0, 1]$ is the crossover probability. In our experiments, CR is set to 0.8 and $rand$ is generated randomly, subject to a uniform distribution.

Selection. In this step, we will compare the fitness of U_n^{t+1} and X_n^t, which is calculated by PSAE whose details are illustrated in Section III.C. If the fitness of U_n^{t+1} is higher than X_n^t, we update X_n^{t+1} with U_n^{t+1}. Otherwise we keep X_n^t unchanged. We formulate selection as:

$$X_n^{t+1} = \begin{cases} U_n^{t+1}, & if \quad PSAE(U_n^{t+1}) > PSAE(X_n^t) \\ X_n^t, & otherwise \end{cases} \tag{6}$$

Re-initialization. Since the initialization is conducted randomly, DE may get stuck in locally optimal solutions. To find the optimal solution robustly, we propose IDE which will re-initialize those individuals who stay unchanged for R generations/iterations.

$$X_n^t = \underset{C \sim p_s}{random}(C), \quad if \quad X_n^t = X_n^{t-1} = ... = X_n^{t-R} \tag{7}$$

where p_s is the distribution of search space. R is set to 3 in our experiments.

Discussion. Compared to DE, the improvements of IDE mainly lie in the re-initialization step. Re-initialization can speed up IDE and avoid IDE getting stuck in locally optimal solutions. To better illustrate the advantages of re-initialization, we utilize DE, **IDE**, EA[5] and ABC[4] to solve the Rastrigin

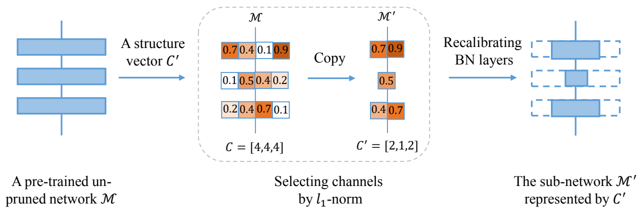
A structure vector C'
$\mathcal{M}$
$\mathcal{M}'$
Copy
Recalibrating BN layers
C = [4,4,4]
C' = [2,1,2]
A pre-trained unpruned network $\mathcal{M}$
Selecting channels by l_1-norm
The sub-network $\mathcal{M}'$ represented by C'

Fig. 4. The pipeline of PSAE. We firstly train an unpruned model $\mathcal{M}$. Then, given a structure vector $C^{'}$, we will estimate its performance by: (1) Sharing weights from $\mathcal{M}$ to $\mathcal{M}^{'}$ by l_1-norm metric. (2) Recalibrating the statistics of BatchNorm layers of $\mathcal{M}^{'}$ with several thousands of samples from the training dataset. Finally, we output the accuracy of $\mathcal{M}^{'}$ on the validation dataset as the estimation of $C^{'}$.

problem[18] which is a popular function to evaluate evolutionary computation:

$$\begin{aligned} & min \quad 10d + \sum_{i=1}^{d} [x_i^2 - 10cos(2\pi x_i)] \\ & s.t. \quad -5 < x_i < 5, i = 1, 2, \ldots, d \end{aligned} \tag{8}$$

Here $d = 30$. As illustrated in Fig.3, IDE can find the optimal solution in less than 100 iterations, while the others cannot find the optimal solution within 500 iterations. This validates the effectiveness of IDE to search for the optimal solution.

C. Pruned Structure Accuracy Evaluator

Prior works are computationally expensive in training a one-shot supernet[5], [3], [2] or fine-tuning the sub-networks[4]. We propose to utilize a pre-trained one-shot unpruned network instead of supernet and fine-tuning in PSAE. As illustrated in Fig.4, PSAE estimates the performance of a given structure vector by two steps: sampling weights from the unpruned model and recalibrating BatchNorm layers.

Suppose an unpruned model $\mathcal{M}$ with a structure vector $C = [c_1, c_2, \cdots, c_L]$, and a pruned model $\mathcal{M}^{'}$ with $C^{'} = [c_1^{'}, c_2^{'}, \cdots, c_L^{'}]$, we have $c_i^{'} < c_i$. For i-th layer of $\mathcal{M}$, we firstly rank all c_i channels by their l_1-norm values, then we select $c_i^{'}$ channels with largest l_1-norm values from $\mathcal{M}$ and copy their weights to $\mathcal{M}^{'}$. It is conducted layer by layer and is inspired by [6].

Since the statistics of BatchNorm layers of $\mathcal{M}^{'}$ are changed, we utilize several thousands of samples to recalibrate BatchNorm layers, like [3]. Note that there is no training in this process and only several thousands of samples from the training dataset are used. Thus calculating BatchNorm post-statistics can be very fast. Finally, PSAE outputs the accuracy of $\mathcal{M}^{'}$ on the validation dataset as the performance estimation of $C^{'}$. Our PSAE lies in the assumption that keeping "important" filters by l_1-norm metric can keep a large part of the information in a model. Detailed analysis is in the next section (Table VI).

IV. Experiments

A. Implementation details

Datasets. We conduct our experiments on both CIFAR10[19] and ImageNet[20] datasets. To make it easier to compare with other methods, we study the performance of AACP on mainstream CNN models, including VGGNet, ResNet and MobileNetV2. Specifically, we prune VGG16/ResNet56/ResNet110 on CIFAR10 and ResNet50/MobileNetV2 on ImageNet.

Training settings. On CIFAR10, our training settings are the same as [9]. We use SGD optimizer and step learning rate scheduler with the momentum of 0.9 and weight decay of 10^{-4}. We train the unpruned model for 160 epochs with an initial learning rate of 0.1 and fine-tune the optimal pruned model for 160 epochs with an initial learning rate of 0.01.

On ImageNet, our data augmentation strategies are the same as PyTorch[21] official examples. For ResNet50 on ImageNet, SGD optimizer and step learning rate scheduler are used. We train the unpruned model for 90 epochs with an initial learning rate of 0.1 and fine-tune the optimal pruned model for 90 epochs with an initial learning rate of 0.01. For MobileNetV2 on ImageNet, we follow the settings in [22]. We use a cosine learning rate scheduler with an initial learning rate of 0.05, momentum of 0.9, weight decay of 0.00004. SGD optimizer is used and the total training epochs is 160.

Pruning settings. We set the population size $N = 10$, iteration number $T = 200$, differential weight $F = 0.5$ and crossover probability $CR = 0.8$, which are evaluated by the Rastrigin problem(Eq.8). After evolutionary search, we fine-tune the optimal structure $C^{'*}$ by default.

B. Visualization of Pruning

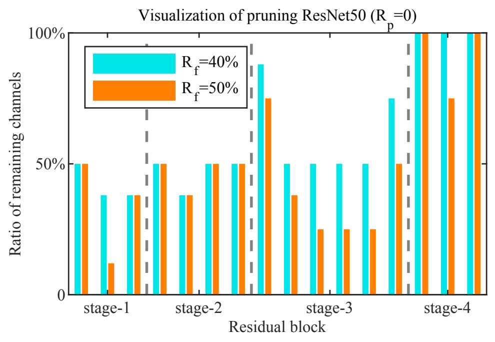

Fig. 5. Visualization of pruning ResNet50 on ImageNet with $R_p = 0$ and $R_f = 40\%/50\%$. ResNet50 has four stages. Each stage has 3 to 6 residual blocks. We draw the proportion of the remaining channels of each block. An unpruned model is 100% reserved for every block. Best viewed in color.

In this part, we visualize our optimal structure vectors and discuss some insights based on the results. We prune ResNet50 on ImageNet with a fixed $R_p = 0$ (no constraint on model size) , while changing R_f from 40% to 50%. The pruned results are illustrated in Fig.5. Firstly, our method tends to prune shallower blocks to reduce FLOPs and keeps more channels in deeper blocks. Secondly, our method chooses to prune more channels of those residual blocks which are in the middle of a stage, while keeping more channels of the first block in each stage. We think this is because the first block of every stage downsamples the feature map and thus requires more channels to avoid information loss.

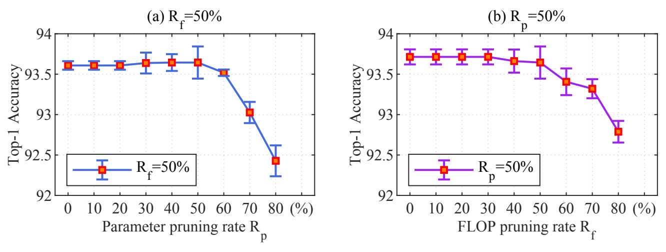

Fig. 6. Visualization of pruning VGG16 on CIFAR10. In (a), we fix $R_f = 50\%$ and change R_p from 0 to 80%. In (b), we fix $R_p = 50\%$ and change R_f from 0 to 80%.

TABLE I
ACCURATE CHANNEL PRUNING OF VGG16 ON CIFAR10 DATASET.

R_f(%)	R_p(%)	r_f(%)	r_p(%)	Acc(%)
50.0	0	50.2	39.8	93.61
50.0	10.0	50.1	40.5	93.61
50.0	20.0	50.1	40.5	93.61
50.0	30.0	50.1	39.4	93.64
50.0	40.0	50.3	40.7	93.64
50.0	50.0	50.1	50.4	93.64
50.0	60.0	57.7	60.0	93.52
50.0	70.0	74.1	70.1	93.03
50.0	80.0	87.1	80.1	92.43
0	50.0	40.3	50.0	93.71
10.0	50.0	40.3	50.0	93.71
20.0	50.0	40.3	50.0	93.71
30.0	50.0	40.3	50.0	93.71
40.0	50.0	40.1	50.1	93.66
50.0	50.0	50.1	50.4	93.64
60.0	50.0	60.0	52.3	93.41
70.0	50.0	70.2	54.9	93.32
80.0	50.0	80.2	55.9	92.79

C. Accurate Channel Pruning

Accurate channel pruning refers to controlling the pruned network in multiple aspects. Without loss of generality, we consider the FLOPs and model size of a pruned network. Inference latency can be handled in the same way because our IDE is a black-box optimization algorithm.

As illustrated in Table I, we firstly fix $R_f = 50\%$ and change R_p from 0 to 80%. When $R_f = 50\%, 0 \leq R_p \leq 50\%$, we can see that $r_f \approx R_f$ and the accuracy of optimal pruned structures does not decrease, which implies that R_f (target of FLOPs) is the stricter constraint compared to R_p (target of parameter). However, when $R_f = 50\%, R_p > 50\%$, we have $r_p \approx R_p$ and the accuracy of optimal pruned structures decrease, which means R_p is the stricter constraint in this case.

Secondly, we fix $R_p = 50\%$ and change R_f from 0 to 80%. The same conclusion can be drawn. These results reveal that the optimal structures are controlled by both R_f and R_p. By changing R_f and R_p, our AACP can accurately compress a model in multiple levels. For better understanding, we illustrate the curve of accuracy in Fig.6.

FLOPs and model size are just two examples to show that our AACP can deal with multiple constraints simultaneously. Other constraints, e.g. inference time, can be applied to AACP as well. Note that these constraints have a strong relationship, but are not linear, as illustrated by Fig.1.

TABLE II
RESULTS ON CIFAR10 DATASET. S REPRESENTS TRAINING $C^{'*}$ FROM SCRATCH AND F REPRESENTS FINE-TUNING $C^{'*}$ BY INHERITING WEIGHTS. $R_p = 0$ FOR ALL RESULTS. R_f IS ANOTATED IN THE TABLE, E.G. (F,50%) MEANS FINE-TUNING $C^{'*}$ AND $R_f = 50\%$.

CNN	Method	ΔFLOPs↓	Baseline(%)	Pruned(%)	ΔAcc(%)
VGG16	L1-norm[6]	34%	93.25	93.40	+0.15
	NS[8]	51%	93.99	93.80	-0.19
	ThiNet[7]	50%	93.99	93.85	-0.14
	CP[23]	50%	93.99	93.67	-0.32
	DCP[24]	50%	93.99	94.16	+0.17
	PFS[22]	50%	93.44	93.63 ±0.06	**+0.19**
	ABCPruner[4]	73%	93.02	93.08	**+0.06**
	AACP(S,50%)	50%	93.56±0.19	93.72±0.13	+0.16
	AACP(F,50%)	50%	93.56±0.19	93.61±0.05	+0.05
	AACP(S,70%)	70%	93.56±0.19	93.57±0.12	+0.01
	AACP(F,70%)	70%	93.56±0.19	93.39±0.07	-0.17
ResNet56	Uniform	50%	92.80	89.80	-3.00
	ThiNet[7]	50%	93.80	92.98	-0.82
	CP[23]	50%	93.80	92.80	-1.00
	DCP[24]	50%	93.80	93.49	-0.31
	AMC[10]	50%	92.80	91.90	-0.90
	SFP[25]	50%	93.59	93.35±0.31	-0.24
	Rethink[1]	50%	93.80	93.07±0.25	-0.73
	PFS[22]	50%	93.23	93.05±0.19	-0.18
	ABCPruner[4]	54%	93.26	93.23	-0.03
	AACP(S,50%)	50%	93.10±0.20	93.31±0.28	**+0.21**
	AACP(F,50%)	50%	93.10±0.20	92.82±0.06	-0.28
ResNet110	L1-norm	40%	93.53	93.30	-0.23
	SFP[25]	40%	93.68	93.86±0.30	+0.18
	Rethink[1]	40%	93.77	93.92±0.13	+0.15
	PFS[22]	40%	93.49	93.69±0.28	+0.20
	ABCPruner[4]	65%	93.50	93.58	+0.08
	AACP(S,40%)	40%	93.30±0.08	93.72±0.43	+0.42
	AACP(F,40%)	40%	93.30±0.08	93.76±0.16	**+0.46**
	AACP(S,65%)	65%	93.30±0.08	93.56±0.12	**+0.26**
	AACP(F,65%)	65%	93.30±0.08	93.33±0.10	+0.03

D. Comparisons with state-of-the-art

In this section, we compare our AACP with other channel pruning methods. To make it fairer, we mainly compare the accuracy drop rate with other methods under the same FLOP pruning rate. We only impose FLOP pruning rate R_f and set $R_p = 0$ in this section, like other channel pruning methods.

We prune VGG16/ResNet56/ResNet110 on CIFAR10 and prune ResNet50/MobileNetV2 on ImageNet. After searching the optimal pruned structure $C^{'*}$, we utilize two ways to resume the accuracy of $C^{'*}$: (1) Fine-tuning $C^{'*}$ by 160 epochs on CIFAR10 and 90 epochs on ImageNet (denoted as F); (2) Training $C^{'*}$ from scratch by 320 epochs on CIFAR10 and 180 epochs on ImageNet (denoted as S).

The results on the CIFAR10 dataset are illustrated in Table II. We run each experiment five times and report the "mean ± std". When pruning VGG16, our method reduces 50% FLOPs of VGG16 with a raise of 0.16% top-1 accuracy, which is only slightly worse than PFS[22] but better than other methods. We also achieve comparable results to ABCPruner[4] when reducing 70% FLOPs of VGG16. When pruning ResNet56 on CIFAR10, our method reduces 50% FLOPs with a raise of 0.21% top-1 accuracy, which surpasses other methods. As for pruning ResNet110 on CIFAR10, we outperform other methods when reducing 40% FLOPs and 65% FLOPs.

We also compare AACP to other channel pruning methods on ImageNet. Table III illustrates the results of pruning ResNet50 and MobileNetV2 on ImageNet. When reducing 42.0% FLOPs of ResNet50, our method only causes 0.06%

TABLE III

RESULTS ON IMAGENET DATASET. TOP-N INDICATES ORIGINAL ACCURACY BEFORE PRUNING. ΔTOP-N INDICATES DEGRADATION OF TOP-N ACCURACY AFTER PRUNING. S REPRESENTS TRAINING $C^{'*}$ FROM SCRATCH AND F REPRESENTS FINE-TUNING $C^{'*}$ BY INHERITING WEIGHTS. $R_p = 0$ FOR ALL RESULTS. R_f IS ANOTATED IN THE TABLE, E.G. (F,42%) MEANS FINE-TUNING $C^{'*}$ AND $R_f = 42\%$. "-" INDICATES THAT NO RESULTS WERE REPORTED IN ORIGINAL PUBLICATION.

CNN	Method	ΔFLOPs↓	Top-1(%)	ΔTop-1(%)	Top-5(%)	ΔTop-5(%)
ResNet50	ThiNet-70[7]	36.8%	72.88	-0.84	91.14	-0.47
	FPGM[9]	42.2%	76.15	-0.56	92.87	-0.24
	DMCP[2]	46.3%	76.60	-0.40	-	-
	HRank[26]	43.8%	76.15	-1.17	92.87	-0.54
	SRR-GR[27]	44.1%	76.13	-0.37	92.86	-0.19
	AACP(S,42%)	42.0%	75.94	-0.46	92.74	-0.18
	AACP(F,42%)	42.0%	75.94	**-0.06**	92.74	**-0.10**
	PFS[22]	51.2%	77.20	-1.60	-	-
	FPGM[9]	53.5%	76.15	-1.32	92.32	-0.55
	AutoSlim[3]	51.2%	76.10	-0.50	-	-
	ThiNet-50[7]	55.8%	72.88	-1.87	90.02	-1.12
	MetaPruning[5]	51.2%	76.60	-1.20	-	-
	ABCPruner[4]	54.3%	76.01	-2.15	92.96	-1.27
	SRR-GR[27]	55.1%	76.13	-1.02	92.86	-0.51
	AACP(S,51%)	51.7%	75.94	-0.93	92.74	-0.63
	AACP(F,51%)	51.7%	75.94	**-0.48**	92.74	**-0.43**
MobileNetV2	Uniform 1.0x	0.0%	71.8	0.0	90.5	0.0
	Uniform 0.75x	30.0%	71.8	-2.5	90.5	-1.7
	PFS[22]	30.0%	72.1	-1.2	-	-
	MetaPruning[5]	27.7%	72.0	-0.8	-	-
	AMC[10]	29.7%	71.8	-1.0	-	-
	AutoSlim[3]	29.7%	74.2	-1.2	-	-
	DMCP[2]	29.7%	74.6	-1.1	-	-
	AACP(S,30%)	30.2%	71.8	-0.7	90.5	-0.8
	AACP(F,30%)	30.2%	71.8	**-0.6**	90.5	**-0.6**

TABLE IV

RESULTS OF PRUNING VGG16/RESNET56 ON CIFAR10 BY RANDOM SELECTION AND $l_1 - norm$ RESPECTIVELY.

Model	metric	ΔFLOPs↓	Baseline(%)	Pruned(%)	ΔAcc(%)
VGG16	random	50.4%	93.56±0.19	93.34±0.15	-0.22
	l_1-norm	50.4%	93.56±0.19	93.61±0.05	**+0.05**
ResNet56	random	50.1%	93.10±0.20	93.21±0.09	+0.11
	l_1-norm	50.1%	93.10±0.20	93.31±0.28	**+0.21**

top-1 accuracy drop and 0.10% top-5 accuracy drop. When reducing 51.7% FLOPs of ResNet50, our method only causes 0.48% top-1 accuracy drop and 0.43% top-5 accuracy drop. For MobileNetV2, we reduce 30.2% FLOPs with 0.6% top-1 and top-5 accuracy degradation. In our experiments, our AACP outperforms other channel pruning methods and achieves state-of-the-art performance. This validates that our method discovers more powerful pruned structures in those cases.

E. Ablation study

Influence of l_1-norm metric. To study the effectiveness of l_1-norm metric used in PSAE, we use random selection to replace l_1-norm and compare their results. We prune VGG16 and ResNet56 on CIFAR10, with l_1-norm and random selection respectively. As Table IV illustrates, l_1-norm metric outperforms random selection. This validates that l_1-norm is an effective criterion to select "important" filters that contain more information of a model when estimating the performance of a given architecture.

The effectiveness of IDE. We have compared our IDE to other evolutionary algorithms in Fig.3. Further, we validate our method on a more challenging problem. We notice that the evolutionary algorithm in MetaPruning[5] was also used in NAS methods[14], [13]. OFA[14] is a two-stage NAS method that firstly trains a once-for-all network by progressive shrinking, then applies evolutionary search to find the optimal structure. The search space includes the width, depth, resolution, and kernel size of a network. As Table V illustrates, we find that replacing the evolutionary algorithm in OFA[14] with our IDE can bring improvements. This result implies that: 1) Our IDE is more effective than the evolutionary algorithm in [5]. 2) Our IDE can be applied to not only channel pruning problems but also general neural architecture search problems.

TABLE V

THE RESULTS OF SEARCHING OFFICIAL PRE-TRAINED ONCE-FOR-ALL NETWORK "OFA_MBV3_D234_E346_K357_W1.2" ON IMAGENET. WITHOUT FINE-TUNING AFTER SEARCHING.

Model	Top-1(%)	FLOPs
OFA w/PS[14]	76.0	230M
OFA w/PS[14]	77.3	300M
OFA w/PS[14]	78.1	399M
OFA w/PS + IDE	76.5	230M
OFA w/PS + IDE	77.6	300M
OFA w/PS + IDE	78.3	399M

TABLE VI

EFFICIENCY COMPARISON WHEN PRUNING RESNET50 ON IMAGENET.

Method	Before Searching		During Searching		Memory Footprint	Total Epochs
	supernet	unpruned network	fine-tune	retrain		
Autoslim[3]	✔	✗	✗	✗	normal	100×4
DMCP[2]	✔	✗	✗	✗	normal	40×4
MetaPruning[5]	✔	✗	✗	✗	huge	32
ABCPruner[4]	✗	✔	✔	✗	normal	$90 + 2NT$
AACP	✗	✔	✗	✗	normal	90

F. Efficiency analysis

In Table VI, we report the total training epochs to find the optimal pruned structures in NAS-based channel pruning methods. A supernet refers to a particularly designed network, which shares weights with its sub-networks[3], [2], [5]. While an unpruned network refers to a standard network such as ResNet and MobileNet. As illustrated in Table VI, Autoslim[3] and DMCP[2] needs more epochs because they adopt the "sandwich rule". MetaPruning[5] trains a PruningNet whose model size is huge thus not efficient. ABCPruner[4] fine-tunes every sub-network for 2 epochs in the searching stage. Overall, our method is more efficient than related methods.

V. CONCLUSION

In this paper, we propose a novel channel pruning method AACP to prune CNNs accurately and automatically, which achieves state-of-the-art performance in many pruning cases. Our method largely speeds up the process of performance estimation and can deal with multiple sparsity constraints to realize accuracy and automatic pruning. In the future, we will explore how to extend the search space and apply AACP to more tasks and networks.

References

[1] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[2] S. Guo, Y. Wang, Q. Li, and J. Yan, "Dmcp: Differentiable markov channel pruning for neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1539–1547.

[3] J. Yu and T. Huang, "Autoslim: Towards one-shot architecture search for channel numbers," *arXiv preprint arXiv:1903.11728*, 2019.

[4] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," *arXiv preprint arXiv:2001.08565*, 2020.

[5] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun, "Metapruning: Meta learning for automatic neural network channel pruning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 3296–3305.

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

[7] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

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

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

[10] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 784–800.

[11] F. Yu, C. Han, P. Wang, R. Huang, X. Huang, and L. Cui, "Hfp: Hardware-aware filter pruning for deep convolutional neural networks acceleration," in *2020 25th International Conference on Pattern Recognition (ICPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jan 2021, pp. 255–262. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICPR48806.2021.9412294

[12] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," 2018.

[13] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *European Conference on Computer Vision*. Springer, 2020, pp. 544–560.

[14] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[15] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017.

[16] Y. Tang, Y. Wang, Y. Xu, Y. Deng, C. Xu, D. Tao, and C. Xu, "Manifold regularized dynamic network pruning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5018–5028.

[17] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[18] "Global optimization test problems," http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm, accessed December 2021.

[19] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[21] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[22] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu, "Pruning from scratch," *arXiv preprint arXiv:1909.12579*, 2019.

[23] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[24] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 875–886.

[25] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," *arXiv preprint arXiv:1808.06866*, 2018.

[26] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1529–1538.

[27] Z. Wang, C. Li, and X. Wang, "Convolutional neural network pruning with structural redundancy reduction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 913–14 922.