BenchCouncil Transactions on Benchmarks, Standards and Evaluations 1 (2021) 100009

Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: https://www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/

Latency-aware automatic CNN channel pruning with GPU runtime analysis

Jiaqiang Liu, Jingwei Sun *, Zhongtian Xu, Guangzhong Sun

School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

ARTICLE INFO

Keywords:
GPU runtime analysis
Inference latency
Channel pruning
Convolutional neural network

ABSTRACT

The huge storage and computation cost of convolutional neural networks (CNN) make them challenging to meet the real-time inference requirement in many applications. Existing channel pruning methods mainly focus on removing unimportant channels in a CNN model based on rule-of-thumb designs, using reduced floating-point operations (FLOPs) and parameter numbers to measure the pruning quality. The inference latency of pruned models is often overlooked. In this paper, we propose a latency-aware automatic CNN channel pruning method (LACP), which aims to search low latency and accurate pruned network structure automatically. We evaluate the inaccuracy of measuring pruning quality by FLOPs and the number of parameters, and use the model inference latency as the direct optimization metric. To bridge model pruning and inference acceleration, we analyze the inference latency of convolutional layers on GPU. Results show that the inference latency of convolutional layers exhibits a staircase pattern along with channel number due to the GPU tail effect. Based on that observation, we greatly shrink the search space of network structures. Then we apply an evolutionary procedure to search a computationally efficient pruned network structure, which reduces the inference latency and maintains the model accuracy. Experiments and comparisons with state-of-the-art methods on three image classification datasets show that our method can achieve better inference acceleration with less accuracy loss.

1. Introduction

Convolutional Neural Networks (CNNs) have demonstrated state-of-the-art achievements in various tasks, such as image classification [1], object detection [2], and image segmentation [3]. Such a success is built upon a large number of model parameters and convolutional operations. As a result, the huge storage and computation cost make these models difficult to be deployed on resource-constrained devices, such as phones and robots. To address this problem, a common approach is to use model compression techniques, including quantization [4], distillation [5], and pruning [6–9]. Among them, neural network pruning has been recognized as one of the most effective tools for compressing CNNs.

Neural network pruning methods aim to remove redundant weights in a dense model. According to the pruning granularity, these methods can be categorized into either weight pruning or channel pruning. In weight pruning, individual weights are zeroed out, leaving a sparse set of weight tensors. Weight pruning can significantly reduce the model size, but it also introduces irregular memory access, leading to very limited or even negative speedups on general-purpose hardware (e.g. CPU, GPU) [10]. Differing from weight pruning, channel pruning methods remove entire channels to compress the model. Since channel pruning only changes the dimension of weight tensors, the pruned model still adopts a dense format, which is well-suited to general-purpose hardware and off-the-shelf libraries. As a result, channel pruning can achieve better acceleration on inference performance than weight pruning.

Due to the promising performance improvement in model compression, channel pruning methods have been widely studied for many years. Existing methods use the reduced floating-point operations (FLOPs) and parameter numbers to measure the pruning quality by default. However, the inference latency of neural network is influenced by many factors, such as the network architecture, the implementation of operators, and the hardware property. Therefore, using FLOPs or the number of parameters as a proxy for inference latency is insufficient, and may lead the algorithm to sub-optimal result. For instance, Fig. 1 shows the relationship between FLOPs, model size, and inference latency of VGG16 network. We randomly prune channels in convolutional layers, then measure the pruned model's FLOPs, number of parameters, and inference latency. Results show that FLOPs or parameter reduction does not necessarily result in latency reduction. For example, the pruned model A has smaller FLOPs than model B, but shows larger inference latency. The same for model C and model D, the smaller model C shows larger inference latency. This observation motivates us to investigate a latency-aware channel pruning method, instead of only focusing on FLOPs or parameter numbers.

* Corresponding author.
E-mail addresses: jqliu42@mail.ustc.edu.cn (J. Liu), sunjw@ustc.edu.cn (J. Sun), xuzt@mail.ustc.edu.cn (Z. Xu), gzsun@ustc.edu.cn (G. Sun).

https://doi.org/10.1016/j.tbench.2021.100009
Received 6 August 2021; Received in revised form 11 October 2021; Accepted 20 October 2021
Available online 3 November 2021

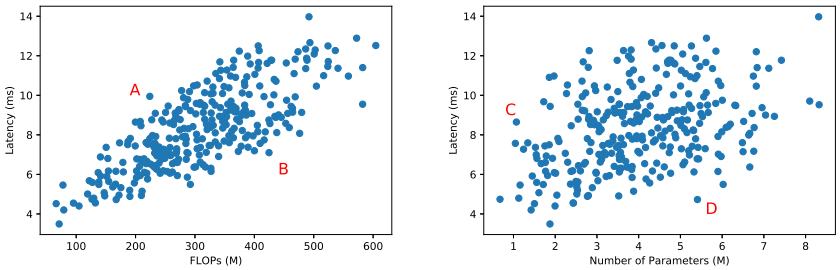

Fig. 1. The relationship between FLOPs, number of parameters, and inference latency of pruned models.

Another motivation of this work is that conventional channel pruning methods crucially rely on human expert knowledge and hand-crafted designs, and focus on selecting unimportant channels. Li et al. [9] take $l1$-norm as significance criteria to determine which channels will be pruned. Luo et al. [11] use the input of $(i+1)$-th layer to guide the pruning of ith layer. Lin et al. [12] rank channels with high rank of feature maps, then prunes the least important channels. However, Liu et al. [13] find that the pruned network can achieve the same accuracy no matter it inherits the weights in the original network or not. This study inspires us that the essence of channel pruning lies in finding optimal channel numbers in each layer, instead of selecting unimportant channels based on rule-of-thumb designs. Following that idea, Lin et al. [14] use artificial bee colony algorithm to search optimal pruned network structure. However, like many conventional channel pruning methods, Lin et al. [14] use the reduced FLOPs and parameter numbers to measure the pruning quality, the latency speedup of pruned model cannot be guaranteed.

In this paper, we propose a latency-aware automatic channel pruning (LACP) method. Differing from conventional methods, we take channel pruning in an automatic manner. Our method aims to search the optimal pruned network structure, i.e., the channel number in convolutional layers, instead of selecting important channels. An intuitive challenge in finding optimal network structure is that it is impractical to exhaustively searching all the possible combinations of pruned network structures. To make the algorithm feasible, effective shrinkage on search space is necessary. We first analyze the inference latency of pruned convolutional layers on GPU. Results show that the inference latency of convolutional layers presents a staircase pattern with the number of channels, which means the inference latency of a convolutional layer changes suddenly at certain channel number intervals. Based on this observation, we greatly shrink the search space of pruned structures. Then we apply an evolutional procedure to efficiently search low-latency and accurate network structure. For each candidate structure, we encode it to a vector $C = [c_1, c_2, c_3, \ldots, c_i]$, where c_i represents the channel numbers in ith convolutional layer. The fitness of candidate pruned network structure is measured in both model accuracy and inference latency. At each population, K candidates with highest fitness will survive to next population, crossover and mutation will take place in these survived structures to generate new structures. Finally, the best candidate is selected as the optimal pruned network structure.

Overall, the main contributions of this paper are as follows:

- We propose a latency-aware automatic channel pruning method LACP. Compared to conventional methods, LACP does not require hand-crafted designs on selecting unimportant channels. It focus on the inference latency speedup, instead of the FLOPs reduction.
- We analyze the inference latency of convolutional layers on GPU. Based on the analysis results, we greatly shrink the search space of pruned network structures, which enables efficient search of low-latency and accurate network structure.
- We conduct a detailed evaluation to compare the proposed method and existing methods on standard datasets. Results show that our method can achieve more latency reduction with less accuracy loss.

The rest of this paper is organized as follows. Section 2 reviews related works. Section 3 presents the proposed latency-aware automatic channel pruning method in detail. Section 4, show the experimental results and analysis. Finally, we draw the paper to a conclusion in Section 5.

2. Related work

Deep neural networks are usually over-parameterized [15,16], leading to huge storage and computation cost. There are extensive studies on compressing and accelerating neural networks. We classify current related research works into two major types: network pruning methods and neural architecture search (NAS) methods.

Pruning methods reduce the storage and computation cost by removing unimportant weights from the origin network. Existing pruning algorithms can be categorized into weight pruning and channel pruning. In weight pruning, individual weights are zeroed out. LeCun et al. [6] present the early work about network pruning using second-order derivatives as the pruning criterion. Han et al. [7] first propose iterative pruning, which prunes individual weights below a monotonically increasing threshold. Guo et al. [17] and Mocanu et al. [18] point out that some previously unimportant weights may tend to be important later. Inspired by this idea, LIU et al. [19] propose a trainable mask-based method to dynamically get sparse network during the training phase. Dettmers and Zettlemoyer [20] propose sparse momentum that used the exponentially smoothed gradients as the criterion for pruning and regrowth. A fixed percentage of parameters are pruned at each pruning step. Weight pruning can significantly reduce the model size. However, the non-structured random connectivity in DNN introduces irregular memory access. It adversely affects practical acceleration in hardware platforms [10]. Differing from weight pruning, channel pruning methods focus on removing the entire redundant channels. Li et al. [9] use $l1$-norm to determine the importance of channels. He et al. [8] formulate channel pruning as an optimization problem, which selects the most representative channels to recover the accuracy of pruned network with minimal reconstruction error. Luo et al. [11] use the next layer's input to guide the pruning of the previous layer. Lin et al. [12] use the feature map rank as sensitivity metric to prune the least important channels. Differing from these magnitude-based or sensitivity-based channel pruning methods, our work performs channel pruning in an automatic manner.

Although network pruning methods have achieved great success, they crucially rely on human expert knowledge and hand-crafted designs. Automatically optimizing the neural network architecture has been widely studied in recent years, known as neural architecture search (NAS). Prior works mainly sample a large number of networks from search space and train them from scratch to obtain a supervision signal, e.g. validation accuracy, for optimizing the sampling agent with reinforcement learning [21–23] or updating the population with an evolutionary algorithm [24]. Bender et al. [25] and Pham et al. [26] introduce weight-sharing paradigm in NAS to boost search efficiency, where all candidate sub-networks share the weights in a single one-shot model that contains every possible architecture in the search space. Liu et al. [27] relax the search space to be continuous with architecture

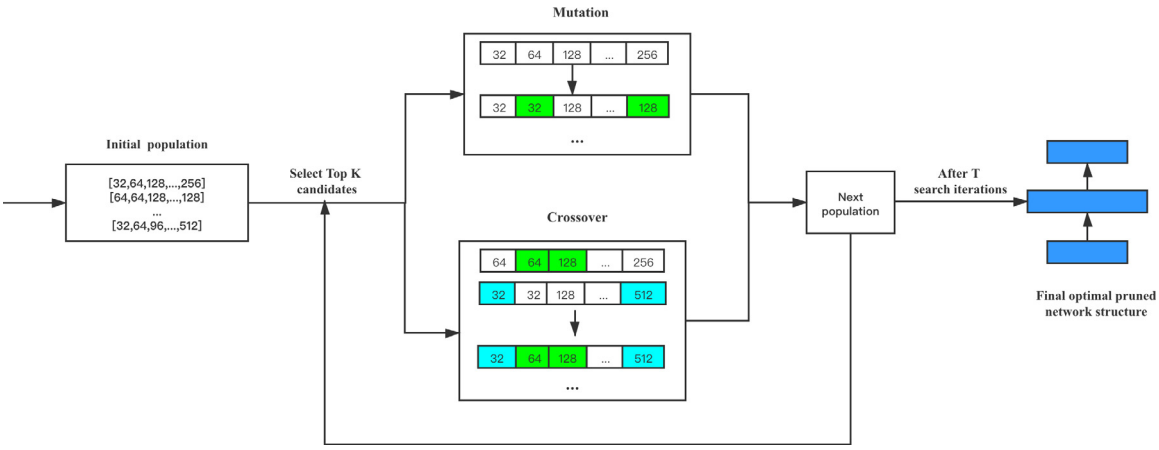

Fig. 2. The overall framework of LACP algorithm.

parameters and then efficiently optimized model parameters and architecture parameters together via gradient descent. Most prior NAS based pruning methods are implemented in a bottom-up and layer-by-layer manner. In contrast, our work mainly focuses on the optimal channel number of each layer.

3. Methodology

3.1. Overview

Cheng et al. [28] point out that convolutional layers take up most of the computation cost in convolutional neural networks. Our work focuses on reducing the channel number of convolutional layers to effectively compress the neural network. Fig. 2 presents the overall framework of our LACP algorithm. Consider a CNN model N that contains L convolutional layers. We refer to $C = [c_1, c_2, \ldots, c_L]$ as the network structure of N, where c_i is the channel number of the ith convolutional layer. We regard channel pruning as an optimal network structure search process, rather than manually designed strategies to remove unimportant channels. The algorithm aims to find a thinner network structure than the unpruned model, meanwhile, keeping a comparable accuracy. We adopt an evolutionary algorithm to achieve the goal of our search algorithm. A certain number of candidate network structures make up a population, the candidate network structures are evaluated using fitness. At each population, the best K candidate network structures are survive to the next population, and those survival candidates will produce new network structures through crossover and mutation. In the end, the best candidate network structure in the whole process is selected to be the optimal pruned network structure, we then fine-tune it to restore the accuracy. Formally, the algorithm is equivalent to solve an optimization problem as Eq. (1) shows.

$$C_{optimal} = \arg\max_{S} \quad F(C, W, D_{train}, D_{test}) \tag{1}$$

S is the search space of pruned network structures. $C \in S$ is the candidate network structure. W is the weight of pruned network, which is assigned from the pre-trained model. D_{train} and D_{test} represent the training data and testing data, respectively. The function F evaluates the fitness of candidate network structure to decide whether keeping current candidate in next population. The effectiveness and efficiency of the search algorithm mostly rely on the fitness evaluation and the search space definition. To find a low-latency and accurate pruned model, the fitness function should consider both model inference latency and test accuracy. For a convolutional neural network that contains L convolutional layers, the possible combination of pruned network structure can be $\prod_{i=1}^{L} c_i$, where c_i represents the channel numbers of ith convolutional layer in original dense model. It is impractical to exhaustively searching all the possible network structures, therefore, effective constraints on the search space are necessary. To solve these problems, we further describe the detailed implementation of our method in the following sections.

3.2. Search space definition

Exhaustively searching every possible pruned network structure is impractical. To make the search algorithm feasible, we need to shrink the search space. In this section, we conduct analysis on inference latency of convolutional layers to find an efficient search space design.

Convolutional layers are widely used in modern neural networks. A convolutional layer consists of a certain number of channels to extract data features. To reduce the computation cost of convolutional layer, channel pruning aims to remove a portion of channels. Intuitively, with the decrease of the channel number, the FLOPs of a convolutional layer will decrease linearly. However, due to the complex nature of convolutional layer's execution environment, its inference latency does not vary linearly with the FLOPs. To better understand the execution mechanism convolutional layer, we analyze how channel pruning affects the inference latency of convolutional layer. As Fig. 3 illustrates, the inference latency of convolutional layers shows a staircase pattern with different number of channels, which means with increasing a certain number of channels, there will be a significant step increase in latency. By analyzing the intrinsic mechanism of DNN deployment on GPU, this phenomenon can be explained. The computation of a convolutional layer is parallelized using multiple threads. These threads are first grouped into different blocks, then loaded to streaming multiprocessors (SMs) on a GPU. The maximum number of blocks loaded on one SM is determined by GPU's physical capacity. If the number of thread blocks in need exceeds the GPU capacity, then GPU will divide these thread blocks into multiple consecutive waves, and run these waves in sequence. Since the SMs are executed in parallel, one wave takes the same amount of time, no matter it is fully occupied or not. This phenomenon is called "GPU tail effect". For different channel number settings of a convolutional layer, their execution time can be very similar if they need the same amount of waves to compute. Therefore, with the increase of channel number, the computation cost of convolutional layer will increase. Once a critical point is exceeded, an extra wave is needed to finish the computation, which leads to a significant step increase in latency. Then, the inference latency of convolutional layer will change slowly, until the last wave is fully occupied.

Inspired by the "GPU tail effect" phenomenon, we can greatly shrink the search space of pruned network structure. Since the inference latency shows a staircase pattern, which means under a certain range

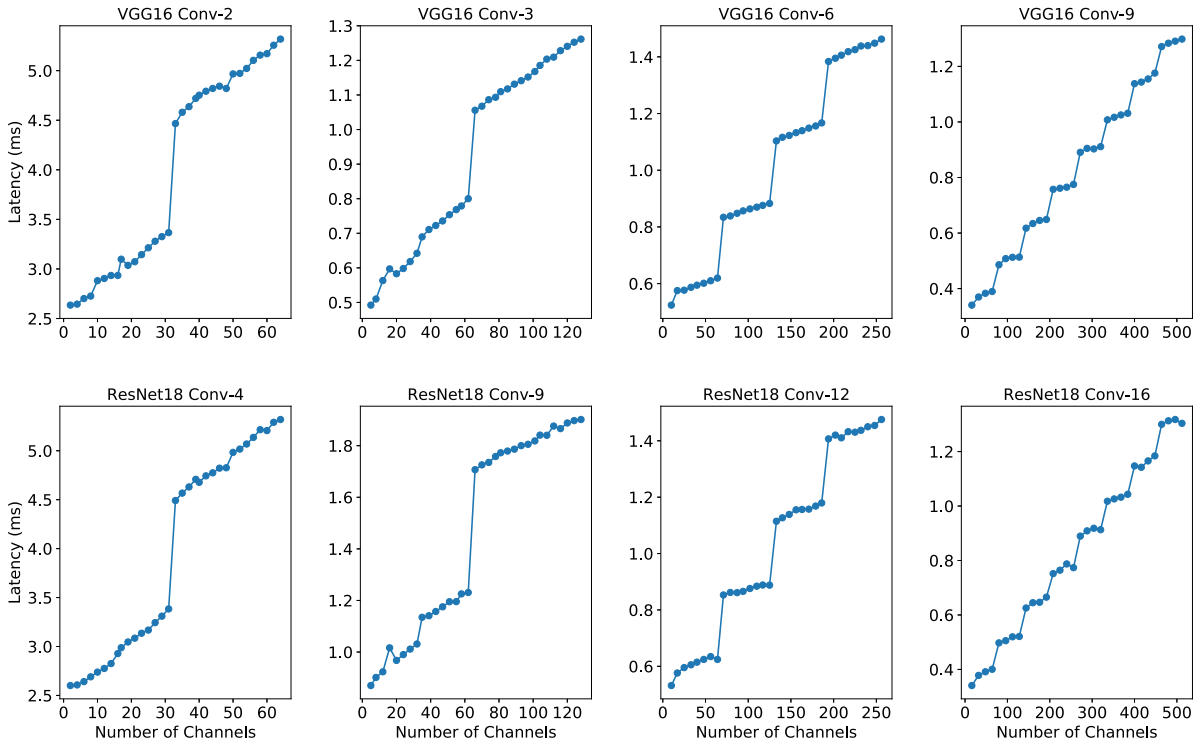

Fig. 3. Inference latency of convolutional layers with varying number of channels.

of channel number settings, the inference latency changes very slowly. Within a staircase step, set the channel number to the right endpoint, then we can maximize the representational capacity of network with a little latency cost.

Algorithm 1 Latency-aware Automatic Channel Pruning Algorithm

Input: Search Cycles: S, Population Size: N, Number of Mutation: M, Number of Crossover: C, Target latency: T
Output: Optimal pruned network structure C^*

G_0 = Random(N)
$G_{topK} = \emptyset$
for $i = 0; i < S; i++$ **do**
  G_{best} = Top1(G_i)
  if G_{best} better than C^* **then**
    $C^* = G_{best}$
  end if
  G_{topK} = TopK(G_i)
  $G_{mutation}$ = Mutation(G_{topK},M)
  $G_{crossover}$ = Crossover(G_{topK},C)
  $G_{i+1} = G_{mutation} + G_{crossover}$
end for
return C^*

We analyze the inference latency variation of different convolutional layers in VGG and ResNet. Results show that the width of the staircase step is a multiple of 32. For the first few convolutional layers, the width of the staircase step is 32. As the layers deepen, the max-pooling operation or the down-sampling operation makes the feature map smaller, thus a single GPU wave can compute more convolution operation. As a result, in the subsequent convolutional layers, the width of the staircase step can increase to 64 or 128. Heuristically, for each convolutional layer, we set its possible channel number in pruned network structure to a multiple of 32. Taking VGG16 as an example, the possible channel number in the sixth convolutional layer is $[32, 64, 96, 128, 160, 192, 224, 256]$, where the initial number of channels is 256. The other convolutional layers are also set up in the same way.

3.3. Optimal network structure search

In this section, we describe the detailed implementation of our LACP method. As Algorithm 1 shows, our method adopts evolutionary search as the overall framework. In the beginning, the initial population is randomly generated from the search space. Each sample in the population represents a pruned network structure, formalized as $C = [c_1, c_2, \ldots, c_L]$, where c_i represents the channel number in ith convolutional layer. At each population, the fitness of every candidate pruned network structure is evaluated as below:

$$fitness(C) = Acc(C) \times \left[\frac{Latency(C)}{T}\right]^w \tag{2}$$

$$w = \begin{cases} 0, & if \quad Latency(C) < T, \\ -1, & otherwise. \end{cases} \tag{3}$$

As Eq. (2) shows, both accuracy and inference latency are considered in the fitness evaluation, where $Acc(C)$ represents the test accuracy of the pruned network. $Latency(C)$ is the inference latency of the pruned network and T is the target latency, which is specified before running the algorithm. To measure the test accuracy of a network structure, it is very time-consuming to completely train and test the pruned model. In our implementation, we initialize the candidate pruned network with the pre-trained model, for a pruned network $C = [c_1, c_2, \ldots, c_L]$, the ith convolutional layer is initialized with c_i channels in the corresponding ith convolutional layer in the pre-trained model, which have larger $l1$-norm value. Then we train the pruned model with 2 epochs and evaluate its test accuracy. Besides, we add a latency constraint in the fitness function. Given a target latency, if the inference latency of pruned network is less than the target latency T, we simply use the test accuracy as the fitness value, otherwise, we penalize the fitness value with a coefficient less than 1. In such a mechanism, the algorithm will tend to select the model whose inference latency reaches the target latency constraint.

At each population, K candidate pruned network structures with largest fitness will survive to next population. Crossover and mutation will take place in these K candidate structures to generate new

structures. The objective of the crossover operation is to integrate excellent information from the parents. For example, given two preserved network structures:

[**32**, **32**, 128, 96, **32**, **192**, 224, 192], [64, 64, **96**, **64**, 160, 96, **320**, **288**]

one new structure will be generated by combining pieces of two parent structures:

[32, 32, 96, 64, 32, 192, 320, 288]

Mutation operation is used to promote population diversity. For example, given a network structure:

[**32**, 32, **128**, 96, **32**, 192, 224, 192]

its fragments are randomly changed, generating a new network structure:

[64, 32, 160, 96, 128, 192, 224, 192]

The preserved K candidate network structures and the structures generated by crossover and mutation form the next population. The algorithm will repeat such search iteration for S times. In the end, the best candidate is selected to be the optimal pruned network structure. We then fine-tune it to restore the accuracy.

4. Evaluation

In this section, we conduct experiments on standard datasets with different models to evaluate the performance of our algorithm.

4.1. Experimental settings

We implement our algorithm with Pytorch 1.5.0. All the experiments are run on NVIDIA GeForce RTX 2080 Ti GPU, which is made up of 4352 CUDA Cores and 68 SMs. We choose three standard image classification datasets (CIFAR-10, CIFAR-100, and Tiny-ImageNet) to evaluate our method. CIFAR-10 dataset consists of 60,000 colored images, which are classified into 10 classes. Each class has 5000 training images and 1000 testing images. Similar to CIFAR-10, CIFAR-100 contains 100 classes of images. Each class has 500 training images and 100 testing images. Tiny-ImageNet contains 100,000 images of 200 classes (500 for each class) colored images. Each class has 500 training images, 50 validation images, and 50 test images.

We use two kinds of models in our experiments: VGG and ResNet. VGG is a single-path network. The 16-layer model is adopted for compression. ResNet consists of a series of blocks, and there is a shortcut between two adjacent blocks. For dimensional matching in the pruned network, the last convolutional layer in each block will not be pruned. Two different depths of ResNet are adopted, including ResNet18 and ResNet34.

For each group of experiments, we report test accuracy, the reduction of network inference latency, the reduction of FLOPs, the reduction of parameter numbers, and the reduction of channel numbers as the performance metrics. We use the PyTorch expansion package thop to count the FLOPs and parameter numbers of network. To measure inference latency of network, we run the model 10 times for GPU warm up, then run the model 300 times with input batch size 128, and take the average inference time.

For each pre-trained model used in our experiments, we train it with 200 epochs using Stochastic Gradient Descent with momentum 0.9, and the batch size is set to 128, the initial learning rate is set to 0.1, which decays by 10 every 50 epochs. The weight decay is set to 1e-4.

4.2. Comparative methods

We compare our method with three representative algorithms to show its effectiveness.

- PFEC [9] is a representative traditional magnitude-based channel pruning method. PFEC calculates and sorts the $l1$-norm value of channels. Channels with smaller $l1$-norm value are less important, then those channels and corresponding feature maps are pruned.
- Thinet [11] formulates channel pruning as an optimization problem, and prunes channels of current layer based on statistics information computed from its next layer.
- ABCPruner [14] is a state-of-the-art automatic channel pruning method. It adopts artificial bee colony algorithm to search optimal pruned network structures. For ith convolutional layer of unpruned model that contains c_i channels, ABCPruner defines its search scope to $\{10\%c_i, 20\%c_i, 30\%c_i, \ldots, \alpha\%c_i\}$, where the maximum preserve percent α is used to restrict the width of pruned network, so that the FLOPs and parameter numbers can be reduced.

4.3. Evaluation results

We conduct our experiments on CIFAR-10, CIFAR-100 and Tiny-ImageNet datasets with VGG and ResNet models. To search for optimal pruned network structures, we set the number of search cycles to 10 and the population size is 30, so LACP searches 300 pruned network structures in the whole process. In each population, the numbers of new pruned network structures that generated from mutation and crossovers are both set to 15. In the end, we fine-tune the best pruned network structure for 200 epochs with a learning rate of 0.1, which is divided by 10 every 50 epochs. The weight decay is set to 1e-4. All algorithms use the same pre-trained model, and the number of fine-tuning epoch is set to 200. For a fair comparison with ABCPruner, we set its maximum searching number of pruned network structures to the same 300.

The experimental results are shown in Table 1, compared with PFEC, Thinet and ABCPruner, our method achieves better model inference acceleration, while maintaining similar or higher accuracy. It is worth noting that, as we have discussed before, more FLOPs or parameter reduction does not necessarily lead to better inference acceleration. Take CIFAR-100 dataset experiments as an example, ABCPruner-50% prunes VGG16 with 87.29% FLOPs reduction and 88.22% parameter reduction, while LACP-0.5 prunes 69.03% FLOPs and 81.14% parameters. However, LACP-0.5 achieves more latency reduction and a significantly higher accuracy than ABCPruner-50%. Another drawback of ABCPruner can be observed from the experimental results. ABCPruner compresses the model by limiting the maximum preserve channel number of convolutional layers. As a result, once the max preserve percent is small, the width of the pruned network is limited and the representational capacity of the pruned model is thus limited. To verify that point, we show a case study in Fig. 4. As shown in the figure, compared with ABCPruner, our method achieves less accuracy loss, while reducing the same percent of inference latency. As a supplementary analysis, we compare the pruned network structure of LACP and ABCPruner, result shows that our method preserves more channels in the first several convolutional layers, which is more important for neural network to extract feature information. On the contrary, the pruned network of ABCPruner has a narrower head structure due to the maximum preserve setting, leading to more accuracy loss.

5. Conclusion

In this paper, we propose a novel latency-aware automatic CNN channel pruning method. Differing from conventional channel pruning methods, our method get rid of selecting unimportant channels based on hand-crafted design, and search for optimal pruned network

Table 1
The experiment results on three datasets for different models. We compare LACP with three other methods.

Dataset	Model	Algorithm	Accuracy (%)	+/- (%)	Latency reduction (%)	FLOPs reduction (%)	Parameter reduction (%)	Channel reduction (%)
CIFAR10	VGG16	dense	93.02	0	0	0	0	0
		PFEC	92.52	−0.5	26.18	47.01	44.57	25
		LACP-0.7	**92.74**	**−0.28**	**33.63**	46.19	53.88	36.36
		Thinet	91.51	−1.51	39.18	61.06	57.88	33.55
		ABCPruner-90%	92.41	−0.61	46.59	68.82	78.39	51.23
		LACP-0.5	**92.62**	**−0.4**	**48.97**	70.08	72.42	50
		ABCPruner-50%	90.11	−2.91	59.18	87.55	87.81	67.57
		LACP-0.4	**91.62**	**−1.4**	**59.56**	85.04	94.46	72.73
	ResNet18	dense	94.28	0	0	0	0	0
		PFEC	92.01	−2.27	17.56	35.26	48.51	18.67
		ABCPruner-90%	94.02	−0.26	0.19	25.44	30.39	12.48
		ABCPruner-50%	93.59	−0.69	18.61	60.27	60.07	24.98
		LACP-0.5	**94.34**	**+0.06**	**22.29**	44.96	69.14	22.33
		Thinet	94.36	+0.08	17.98	37.84	37.51	15.29
		LACP-0.7	**94.37**	**+0.09**	**21.3**	41.15	61.71	20.67
CIFAR100	VGG16	dense	69.78	0	0	0	0	0
		PFEC	69.45	−0.33	26.14	47	44.43	25
		LACP-0.7	**70.54**	**+0.76**	**30.23**	50.72	63.17	39.39
		Thinet	69.32	−0.46	40.63	63.42	60.01	35.27
		ABCPruner-90%	69.06	−0.72	28.82	54.11	72.41	41.5
		ABCPruner-50%	65.95	−3.93	39.14	87.29	88.22	66.17
		LACP-0.5	**69.42**	**−0.36**	**49.4**	69.03	81.14	55.3
	ResNet34	dense	74.86	0	0	0	0	0
		PFEC	69.52	−5.34	20.91	39.63	48.93	21.05
		Thinet	74.59	−0.27	19.94	38.09	37.75	16.96
		ABCPruner-90%	74.58	−0.28	25.05	63.65	61.15	27.1
		ABCPruner-50%	74.18	−0.68	23.56	67.92	68.33	30.06
		LACP-0.5	**74.59**	**−0.27**	**27.91**	56.91	69.94	29.32
Tiny-ImageNet	ResNet18	dense	57.87	0	0	0	0	0
		PFEC	56.49	−1.38	17.68	35.26	48.09	18.67
		Thinet	56.53	−1.34	17.81	37.45	37.19	15.29
		ABCPruner-90%	56.55	−1.32	2.69	38.9	34.37	15.27
		ABCPruner-50%	55.23	−2.64	22.35	67.94	71.77	28.08
		LACP-0.7	**56.87**	**−1**	**22.15**	50.16	68.08	24.67
	ResNet34	dense	59.19	0	0	0	0	0
		PFEC	58.98	−0.21	21.24	39.64	48.81	21.05
		Thinet	59.05	−0.14	18.57	37.91	37.66	16.96
		LACP-0.7	**59.02**	**−0.17**	**24.45**	52.28	67.1	27.07
		ABCPruner-90%	58.58	−0.61	10.44	42.21	52.34	20.84
		ABCPruner-50%	57.96	−1.23	24.17	68.09	73.76	31.58
		LACP-0.5	**58.64**	**−0.55**	**27.12**	61.89	76.58	31.58

Note: LACP-α means we set the target latency to $\alpha \times L$, where L is the unpruned model's inference latency. ABCPruner-β means the maximal preserved channel number in each convolutional layer is $\beta \times C$, where C is the original channel number in that layer.

structure automatically. By analyzing the inference latency of pruned networks, we indicate that neither FLOPs nor the number of parameters can accurately represent the real inference acceleration. Besides, we analyze the execution mechanism of convolutional layers on GPU. Results show that the inference latency of convolutional layers presents a staircase pattern with different number of channels. Based on this observation, we greatly shrink the combinations of network structure, enabling efficient search of low-latency and accurate pruned network. We conduct extensive evaluations to compare our method with existing studies on public datasets, and report the real latency metric. Experimental results show that our method can achieve better inference acceleration, while maintaining higher accuracy.

Although we have achieved desired pruning effect on our experiments, our method can be further improved. As we discussed before,

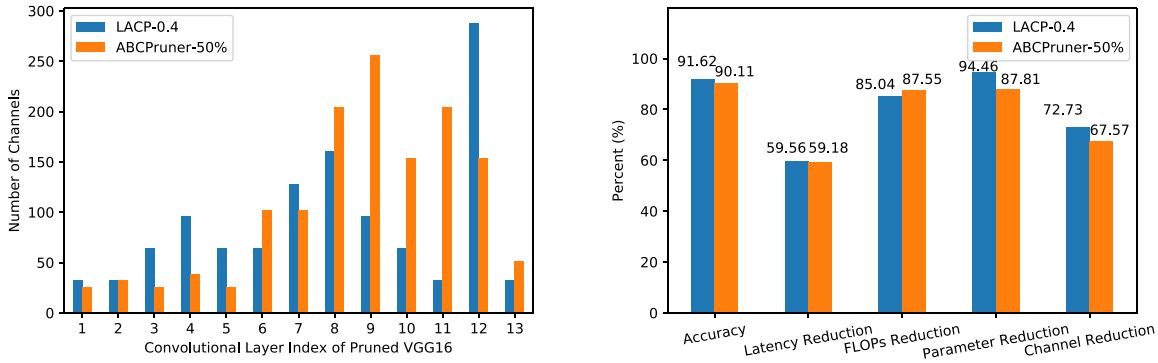

Fig. 4. The pruning results of LACP and ABCPruner for VGG16 on CIFAR-10 dataset.

we shrink the search space of pruned network structure through the analysis of the GPU tail effect. However, our analysis is based on empirical profiling. A more thorough and general investigation of the GPU tail effect could be helpful. Besides, how to generalize our method to different hardware platforms is also worth studying in future work.

Acknowledgments

This work is supported by National Natural Science Foundation of China (No. 61772485). It is also funded by Youth Innovation Promotion Association of Chinese Academy of Sciences (CAS) and JD AI research. Experiments in this study were conducted on the supercomputer system in the Supercomputing Center of University of Science and Technology of China.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[2] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580–587.

[3] Jonathan Long, Evan Shelhamer, Trevor Darrell, Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431–3440.

[4] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, Yoshua Bengio, Quantized neural networks: Training neural networks with low precision weights and activations, J. Mach. Learn. Res. 18 (1) (2017) 6869–6898.

[5] Chenglin Yang, Lingxi Xie, Chi Su, Alan L. Yuille, Snapshot distillation: Teacher-student optimization in one generation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 2859–2868.

[6] Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, Lawrence D Jackel, Optimal brain damage, in: NIPs, Vol. 2, Citeseer, 1989, pp. 598–605.

[7] Song Han, Jeff Pool, John Tran, William Dally, Learning both weights and connections for efficient neural network, in: C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 28, Curran Associates, Inc., 2015.

[8] Yihui He, Xiangyu Zhang, Jian Sun, Channel pruning for accelerating very deep neural networks, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1389–1397.

[9] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, Hans Peter Graf, Pruning filters for efficient convnets, 2016, arXiv preprint arXiv:1608.08710.

[10] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, Hai Li, Learning structured sparsity in deep neural networks, in: Proceedings of the 30th International Conference on Neural Information Processing Systems, 2016, pp. 2082–2090.

[11] Jian-Hao Luo, Jianxin Wu, Weiyao 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.

[12] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, Ling 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.

[13] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, Trevor Darrell, Rethinking the Value of Network Pruning, in: International Conference on Learning Representations, 2018.

[14] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, Yonghong Tian, Channel Pruning via Automatic Structure Search, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 2020, pp. 673–679.

[15] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, Rob Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in: Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 1, 2014, pp. 1269–1277.

[16] Jimmy Ba, Rich Caruana, Do deep nets really need to be deep? in: NIPS, 2014.

[17] Yiwen Guo, Anbang Yao, Yurong Chen, Dynamic network surgery for efficient DNNs, in: NIPS, 2016.

[18] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, Antonio Liotta, Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science, Nature Commun. 9 (1) (2018) 1–12.

[19] Junjie LIU, Zhe XU, Runbin SHI, Ray C. C. Cheung, Hayden K.H. So, Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers, in: International Conference on Learning Representations, 2020.

[20] Tim Dettmers, Luke Zettlemoyer, Sparse networks from scratch: Faster training without losing performance, 2019, arXiv preprint arXiv:1907.04840.

[21] Bowen Baker, Otkrist Gupta, Nikhil Naik, Ramesh Raskar, Designing neural network architectures using reinforcement learning, in: International Conference on Learning Representations, 2018.

[22] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V Le, Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8697–8710.

[23] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, Quoc V Le, Mnasnet: Platform-aware neural architecture search for mobile, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 2820–2828.

[24] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, Alexey Kurakin, Large-scale evolution of image classifiers, in: International Conference on Machine Learning, PMLR, 2017, pp. 2902–2911.

[25] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, Quoc Le, Understanding and simplifying one-shot architecture search, in: International Conference on Machine Learning, PMLR, 2018, pp. 550–559.

[26] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, Jeff Dean, Efficient neural architecture search via parameters sharing, in: International Conference on Machine Learning, PMLR, 2018, pp. 4095–4104.

[27] Hanxiao Liu, Karen Simonyan, Yiming Yang, Darts: Differentiable architecture search, in: International Conference on Learning Representations, 2018.

[28] Jian Cheng, Pei-song Wang, Gang Li, Qing-hao Hu, Han-qing Lu, Recent advances in efficient computation of deep convolutional neural networks, Front. Inf. Technol. Electron. Eng. 19 (1) (2018) 64–77.