Beyond Network Pruning: a Joint Search-and-Training Approach

Xiaotong Lu[1] , **Han Huang**[1] , **Weisheng Dong**[*1] , **Xin Li**[2] , **Guangming Shi**[1]
[1]Xidian University
[2]West Virginia University
{xiaotonglu47, hanhuang8264}@gmail.com, wsdong@mail.xidian.edu.cn, xin.li@ieee.org, gmshi@xidian.edu.cn

Abstract

Network pruning has been proposed as a remedy for alleviating the over-parameterization problem of deep neural networks. However, its value has been recently challenged especially from the perspective of neural architecture search (NAS). We challenge the conventional wisdom of pruning-after-training by proposing a joint search-and-training approach that directly learns a compact network from the scratch. By treating pruning as a search strategy, we present two new insights in this paper: 1) it is possible to expand the search space of networking pruning by associating each filter with a learnable weight; 2) joint search-and-training can be conducted iteratively to maximize the learning efficiency. More specifically, we propose a coarse-to-fine tuning strategy to iteratively sample and update compact sub-network to approximate the target network. The weights associated with network filters will be accordingly updated by joint search-and-training to reflect learned knowledge in NAS space. Moreover, we introduce strategies of random perturbation (inspired by Monte Carlo) and flexible thresholding (inspired by Reinforcement Learning) to adjust the weight and size of each layer. Extensive experiments on ResNet and VGGNet demonstrate the superior performance of our proposed method on popular datasets including CIFAR10, CIFAR100 and ImageNet.

1 Introduction

Network pruning has been a popular remedy for the over-parameterization problem of deep neural networks [Liu *et al.*2018b]. A typical procedure of network pruning consists of three stages: $train$, $prune$, and $fine\text{-}tune$. Traditional approaches implement the strategy of pruning unimportant network structures by applying various heuristics such as sparsity regularization [Lin *et al.*2019], low-rank approximation and weight quantization. There are two common beliefs about network pruning: 1) the necessity of training a large, over-parameterized network (as the target for network pruning); and 2) the importance of preserving both pruned architecture and its associated weights.

It turns out that both believes are not necessarily true for several reasons [Frankle and Carbin2018, Liu *et al.*2018b, Dettmers and Zettlemoyer2019, Gu *et al.*2018]. First, so-called lottery ticket hypothesis [Frankle and Carbin2018] claims that there exist smaller subnetworks that can reach at least similar performance (accuracy and efficiency) to their larger counterparts. Second, structured pruning techniques can often automatically discover compact network architectures from the scratch and without preserving the weights. Inspired by these latest findings, a flurry of recent works have explored novel ways of reducing the computational cost of deep neural networks in low-resource settings such as sparse momentum [Dettmers and Zettlemoyer2019], early-bird tickets [You *et al.*2019], and regularized evolution [Real *et al.*2019].

By casting pruning as a special case of neural architecture search (NAS), we advocate a novel *joint search-and-training* framework for learning efficient compact networks. Inspired by the latest advances in architecture engineering, we introduce two new insights to the field in this paper. First, we propose to expand the search space of networking pruning by associating each filter with a learnable weight. Similar to Transformable Architecture Search(TAS) [Dong and Yang2019a], we find a compact network by calculating the distribution of corresponding weights. Second, we propose to iteratively update the network architecture in a coarse-to-fine fashion to maximize the learning efficiency (conceptually similar to progressive NAS). Combining the strategy of NAS with iterative training, it is possible to obtain highly competitive compact networks from the scratch for only a few hours instead of days.

Under the proposed joint search-and-training framework, we have developed a new algorithm for efficiently learning compact networks (refer to **Figure 1**). In our approach, we alternate between two phases: sampler and updater. In sampling stage, we search for a compact network by weight calculation and thresholding; in the updating stage, the parameters and weights of trained compact network are mapped back to the target network. Meantime, strategies of random perturbation and flexible thresholding are introduced to further improve the efficiency of sampler [Dong and Yang2019c]. Af-

*Corresponding author

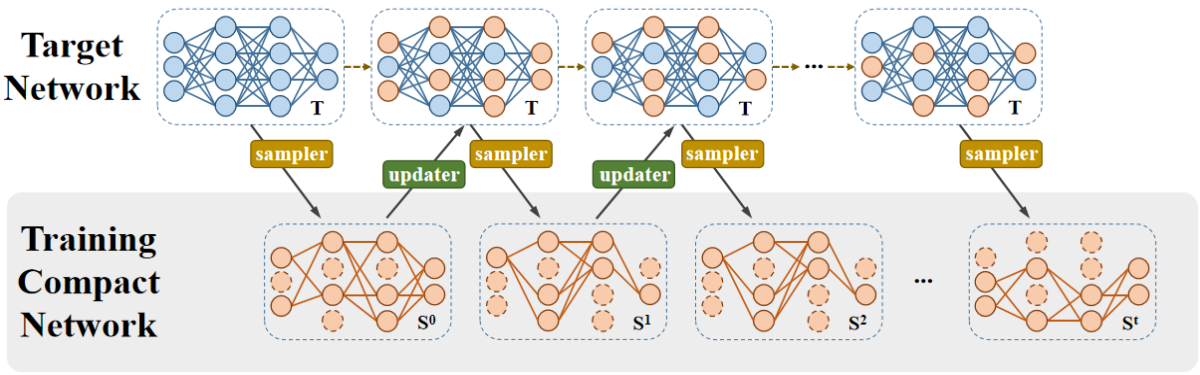

Figure 1: Overview of our joint search-and-training approach. The 'sampler' searches for compact sub-networks from the target network, while the 'updater' maps the trained sub-networks back to the target network. The best performing sub-network will be further fine-tuned as the final output.

ter iterative search-and-training, we select the best compact networks and fine-tune them to convergence. We have validated the proposed method on both ResNet and VGGNet and several well-known datasets (CIFAR-10, CIFAR-100 and ImageNet). Our experimental results show that our method can achieve better accuracy results and shorter wall-clock running time when compared to other competing methods.

2 Related Works

Unstructured Network Pruning. Unstructured Pruning methods aim to emphasize the sparseness of filters or channels in large networks. In this way, according to the sparse theory, the pruned network after trimming the sparse terms will have distribution similar to that of large networks without pruning. To optimize the number of neurons in a network, thus the number of parameters, [Zhou *et al.*2016] incorporates a sparse constraint into the objective function and decimate the number of neurons during the training stage. [Alvarez and Salzmann2016] propose to apply group sparsity regularization on the parameters of the network, where each group is defined to act on a single neuron. Similar to our algorithm, an efficient neural network pruning algorithm is proposed in [Kim *et al.*2019], which can prune the network in a few minutes. However, due to manual design, this method has no advantage in performance and also relies on the well-trained target network.

Structured Network Pruning. Structured Pruning methods iteratively prune and tune filters or layers in the network to reduce the performance damage of hand-crafted design. A typical pruning paradigm is to first train a large target network, then pruning, and finally fine-tune the compact network. The pruning regulation is to prune the relatively unimportant filters and combine the remaining filters into a new compact network. The mainstream pruning methods focus on how to prune properly under the framework of training before pruning. For example, in [He *et al.*2019], the redundancy of the filter is determined by introducing geometric median; [He *et al.*2018a] additionally updates the pruned filter to increase the capacity of the model; [Dong and Yang2019a] applies NAS to search the optimal depth and width of the network; [Liu *et al.*2019] automatically prune networks by meta-learning. In contrast to previous pruning methods, our method allows direct learning of a compact network through joint search and training in the unpruned network, thereby greatly improving the efficiency.

Lightweight Network Designing. Lightweight Network Designing methods avoid pruning by constructing a simpler network structure than standard networks, while achieving the goal of reducing model parameters and calculations. The most popular lightweight network design methods are MobileNet and ShuffleNet. MobileNet achieves network lightweighting by replacing a standard convolution with a depth-wise separable convolution , which contains a depth-wise convolution and a 1×1 convolution. ShuffleNet architecture uses two operations: pointwise group convolution to help reduce computational complexity, and channel shuffle to help information flow. For a given computational complexity budget, ShuffleNet allows more feature maps to be used to help encode more information on small networks.

Network Architecture Search. Network Architecture Search methods aim to find the potentially optimal network structure from hyper-parameterized networks, but usually requires a lot of computing resources and time. Especially for methods based on evolutionary algorithms (EA) [Real *et al.*2019] and reinforcement learning (RL) [Pham *et al.*2018, Zoph *et al.*2018], although those algorithms have made some efforts to improve efficiency. Gradient-based methods [Liu *et al.*2018a] can effectively reduce the calculation cost. However, in order to make the error back propagation, the number of filters in all candidates is fixed, which also leads to the bloated target network. In [Dong and Yang2019b, Cai *et al.*2018], the algorithm based on one-shot learning greatly speeds up the training process with parameter sharing, but training a over-parameterized network is still a heavy burden.

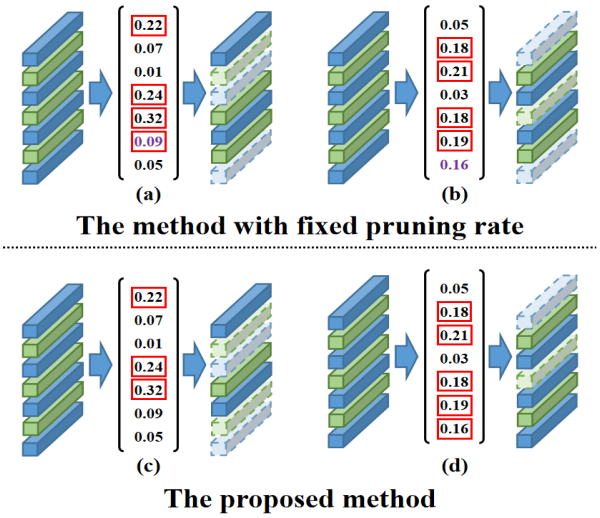

Figure 2: Comparison with method with fixed pruning rate.The number in the figure represent the weight of the corresponding filter, and the ones framed in red are reserved. In (a) and (b), inappropriate pruning occurs (highlighted by purple color in weights) due to a fixed pruning rate: filter with small weight is retained and the one with large weight is discarded. In our method, we tend to consider the distribution of weights as a whole and prune them more adaptively.

3 Efficient Compact Network Learning

3.1 Overview

Different from network pruning, our method does not rely on a pre-trained and over-parametrized model but directly searches and learns a compact network from the scratch. As shown in **Figure 1**, our approach tackles the problem of joint search-and-training in a coarse-to-fine manner by alternating between two stages: *sampler and* updater. The sampler searches for compact sub-networks from the target network, while the updater maps the trained sub-networks back to the target network. The best performing sub-network will be further fine-tuned as the final output. Inspired by Monte Carlo and Reinforcement Learning, we count on random perturbation and iterative thresholding to efficiently search a compact network during the sampler stage.

At the starting point, we first need to train the target network coarsely before searching. Similar to previous works [Ye *et al.*2018], we introduce the concept of setting a flexible threshold for the pruned filter, which requires that the weight can correspond to the importance of the corresponding filter. For this reason, we have added an additional constraint on the weight distribution as a term of training errors in addition to the cross-entropy classification loss. At the initialization, the complete training loss function ℓ_{coarse} to coarsely train the target network $\mathcal{T}$ is given by:

$$\begin{aligned}\ell_{coarse} =& CrossEntropyLoss(y_i, \mathcal{T}(x_i, \Omega, \alpha)) \\ &+ \lambda \left\|\alpha\right\|_2\end{aligned} \tag{1}$$

where (x_i, y_i) denote the pair of input/output training data, α is the weight associated with each filter, Ω refers to parameters in the target network $\mathcal{T}$, $\|\cdot\|_2$ refers to the ℓ_2 norm regularization of the weights and λ is the hyper-parameter.

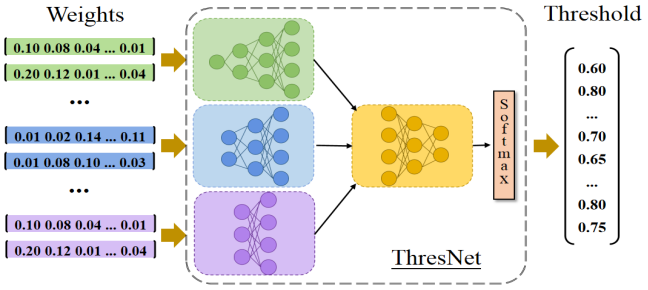

Figure 3: The structure of our proposed ThresNet. The ThresNet is totally consist of fully connected layers, and for network weights input with different size, we process them with different fully connected layers. The final output is the pruning threshold of the corresponding input layer.

After coarsely training the target network, we continue to iteratively search and obtain a series of compact networks $S^0, S^1, \ldots, S^T$ as shown by the sampler in **Figure 1**. These networks will be mapped back to the target network after proper training, which can be interpreted as pruning-based NAS in a coarse-to-fine manner. After the search, we choose a compact network that performs the best on the test set and fine-tune it until reaching convergence. At the core of each sampler is a Threshold network, which plays the role of adapting the thresholds of input layers for network pruning.

3.2 Flexible Pruning via ThresNet

Filters pruning operations are the soul of networking pruning methods because the filter kept at each layer will affect the accuracy of pruned network. Therefore, the pruning operation needs to strike the optimal tradeoff between performance (e.g., accuracy) and cost (e.g., drop rate). Previous filter-wise pruning methods manually control the number of pruned filters. Other filter pruning methods (e.g., [Dong and Yang2019a]) search for the depth and width of a small network and transfer the knowledge from the trained large network to the searched small network. However, the optimality of those filter pruning for a fixed number or drop rate is questionable. For example, when the number of filters selected for each layer is fixed (as shown by the top diagram in **Figure 2**), it may be the case that some large and significant weights get pruned or some small and insignificant weights get reserved in order to meet the strict pruning rate constraint.

It is conceptually desirable to prune the filters more adaptively - e.g., via thresholding the weights (as shown by the bottom diagram in **Figure 2**). Inspired by [He *et al.*2018b] that used reinforcement learning to effectively sample the space of NAS, we propose to train a network called ThresNet to automatically learn the threshold for each layer of the network. We formulate the threshold learning as a sequential decision making process, where an agent - ThresNet parameterized by Φ - can make a sequence of decisions about the threshold β^l according to the filter weight α^l.

$$\beta^l = \Phi\left(\alpha^l\right) \tag{2}$$

The ThresNet takes α^l of each convolutional layer as the input and selects a threshold in the range (e.g., [0.6,0.8] as shown in **Figure 3**) for each layer as the output decision. This network consists of two parts. The first part is an expansion network decomposed of fully connected layers with varying shape sizes corresponding to different shapes of α^l . For example, the varying shape sizes of α^l are 16/32/64 respectively in ResNet for the CIFAR dataset. We design a fully connected network with 3 layers/2 layers/1 layer separately in order to make the input shape size (128) consistent. The second part is a shrinking network that reduces the feature dimension from 128 to the shape size in the threshold space and pick the appropriate threshold as the network output.

As the searching process for compact networks goes, the forward propagation loss of each $\mathcal{S}$ is first calculated. The opposite of the average of losses in every K episodes is then used as a part of the reward (as in reinforcement learning). In order to avoid finding too large sub-networks, the parameter amount is counted as a restriction and the opposite of it is also used as a part of the reward.

$$R = -\left(l_{avg} + \gamma \times param\left(S\right)\right)) \tag{3}$$

Where R refers to the reward, l_{avg} is the average of losses, γ is the hyper-parameter that balance the two parts of the reward and $param\left(S\right)$ denotes the parameter amount. We opt to train the ThresNet using a policy gradient algorithm to learn the policy π_Φ with parameters Φ by maximizing the rewards. The policy gradient can be approximated by [Ashok *et al.*2017]

$$\nabla_\Phi J(\Phi) \approx \frac{1}{K}\sum_{k=1}^{K}\sum_{l=1}^{L} R_k \nabla_\Phi \log \pi_\Phi\left(\beta^l|\alpha^l\right) \tag{4}$$

where R_k is the reward computed at the k-th episode and β^l is the threshold agent decided at layer l. Training ThresNet makes it possible to adaptively learn the structure of compact sub-networks, which brings more flexibility to our NAS method. The sequence of thresholds learned by ThreshNet will be exploited to search for a compact sub-network $\mathcal{S}$ next.

3.3 Sampler: Search the Compact Subnetwork

Given a target Network $\mathcal{T}$ with L layers, we denote all convolution filters in the l-th layer by $W_1^l, W_2^l, \cdots, W_{\mathbb{C}_l}^l \in \mathbb{R}^{\mathbb{C}_{l-1}\times k\times k}$, where $\mathbb{C}_l$ indicates the channel depth of the l-th layer and k represents the kernel size. Therefore, for a given input feature map $\mathbb{I}^l \in \mathbb{R}^{C_{l-1}\times H_{l-1}\times W_{l-1}}$, the output of the l-th layer $\mathbb{O}^l$ can be expressed as:

$$\mathbb{O}^l = \textit{Concat}(\mathbb{I}^l * W_1^l, \mathbb{I}^l * W_2^l, \cdots, \mathbb{I}^l * W_{\mathbb{C}_l}^l) \tag{5}$$

where $*$ indicates the convolution operation. Note that each channel of the output in this layer can be regarded as the convolutional output of the input and the corresponding convolution filter separately. The goal of network pruning is to eliminate unimportant filters for the purpose of reducing the amount of parameters and calculations from the target network. In order to characterize the importance of each convolution filter more directly, we introduce a filter-related weight $\alpha^l \in \mathbb{R}^{C_l}$ into those filters and rewrite Eq. (5) as:

$$\mathbb{O}^l = \textit{Concat}(\mathbb{I}^l * \hat{\alpha}_1^l W_1^l, \mathbb{I}^l * \hat{\alpha}_2^l W_2^l, \cdots, \mathbb{I}^l * \hat{\alpha}_{\mathbb{C}_l}^l W_{\mathbb{C}_l}^l)$$
$$s.t. \quad \hat{\alpha}_c^l = \frac{exp(\alpha_c^l)}{\sum_{k=1}^{\mathbb{C}_l} exp(\alpha_k^l)} \tag{6}$$

Taking filters and their associated weights α^l as inputs, sampler outputs a compact subnetwork with pruned filters and updated weights p^l. Two optimization strategies are introduced to our sampler: 1) we propose to expand the search space by exploiting the randomness - i.e., filter-related weights are randomly perturbed by uniform noise before calculating the probabilities; 2) we propose to make the thresholds flexible with respect to each layer by leveraging the power of ThreshNet in the previous subsection. When combined together, these two strategies can be interpreted as NAS based on Monte Carlo Search [Dai *et al.*2019] and Reinforcement Learning [Zoph *et al.*2018]. The detailed procedure for Sampler is shown in Algorithm 1 where we elaborate on how to choose pruned weights p^l and filters $W_{1,2,\ldots,\mathbb{F}_l}^l$ for the l-th layer based on input weights.

Algorithm 1: Sampler

Input: α^l and $W_{1,2,\ldots,\mathbb{C}_l}^l$

Add uniform noise on α :
$n^l = \alpha^l - log(log(u))\ \ s.t.\ u \in \mathbf{U}(0,1)$;

Calculate the probability of the j-th candidate :
$\hat{n}_j^l = \frac{exp(n_j^l)}{\sum_{k=1}^{\mathbb{C}_l} exp(n_k^l)}$;

Calculate the threshold for l-th layer's weigh :
$th^l = \textbf{ThresNet}(\alpha^l)$;

for $c = 1; s < th^l; c++$ **do**

  Calculate the sum of the largest c candidates in $\hat{n}^l$: $s = sum(\textbf{Top}(\hat{n}^l, c))$;

$p^l = \textbf{Top}(\alpha^l, c)$;

$W_{1,2,\ldots,\mathbb{F}_l}^l = \textbf{Top}(W_{1,2,\ldots,\mathbb{C}_l}^l, c)$;

Output: p^l and $W_{1,2,\ldots,\mathbb{F}_l}^l$

3.4 Joint Search-and-Training

For a searched compact network, the output of each layer can be written as:

$$\hat{\mathbb{O}}^l = \textit{Concat}(\hat{\mathbb{I}}^l * \hat{p}_1^l W_1^l, \hat{\mathbb{I}}^l * \hat{p}_2^l W_2^l, \cdots, \hat{\mathbb{I}}^l * \hat{p}_{\mathbb{F}_l}^l W_{\mathbb{F}_l}^l)$$
$$s.t. \quad \hat{p}_c^l = \frac{exp(p_c^l)}{\sum_{k=1}^{\mathbb{F}_l} exp(p_k^l)} \tag{7}$$

where p^l is the output of **Algorithm 1** and $\hat{p}_c^l$ indicates the normalized weights for each layer. By sampling each layer of the network, we can get a compact sub-network $\mathcal{S}$ of the

Algorithm 2: Search-and-Training Algorithm

Input: The whole available training set $\mathbb{D}$

Initialize the target network $\mathcal{T}$ with filter-related weight α using Eq. (6);
Split the whole training set $\mathbb{D}$ into $\mathbb{D}_{train}$ and $\mathbb{D}_{valid}$;
Coarsely train the target network $\mathcal{T}$ on $\mathbb{D}$ by Eq. (1);
for $t = 1 \rightarrow T$ **do**
  Sample a sub-network $\mathcal{S}^t(\Theta, p) \subset \mathcal{T}$ based on **Algorithm 1**;
  for $m = 1 \rightarrow M$ **do**
    Sample train batch $\mathbb{B}_t = \{(x_i, y_i)\}_{i=1}^{batch}$ from $\mathbb{D}_{train}$
    Optimize Θ on the $\mathbb{B}_t$ by Eq. (8)
  for $n = 1 \rightarrow N$ **do**
    Sample valid batch $\mathbb{B}_v = \{(x_i, y_i)\}_{i=1}^{batch}$ from $\mathbb{D}_{valid}$
    Optimize p on the $\mathbb{B}_v$ by Eq. (9)
  Update $\mathcal{T}$ via optimized Θ and α;
Pick up the best performing network $\mathcal{S}$ and fine-tune it on $\mathbb{D}$ until convergence;

Output: A compact network $\mathcal{S}$

target network $\mathcal{T}$, whose parameters and weights can be expressed by Θ and p, respectively. By feeding these information back to the resampler though the updater as shown in **Figure 1**, we observe that the search for the most compact subnetwork can be solved iteratively in a coarse-to-fine manner.

Just like the NAS method [Zoph *et al.*2018], we can update the parameters and weights on the training set $\mathbb{D}_{train}$ and validation sets $\mathbb{D}_{valid}$ successively, which can be formulated as:

$$\Theta = \underset{\Theta}{\operatorname{argmin}} \sum_{i=1}^{\mathbb{B}_t} \ell_{train}(y_i, \mathcal{S}(x_i, \Theta, p)) \tag{8}$$

$$p = \underset{p}{\operatorname{argmin}} \sum_{i=1}^{\mathbb{B}_v} \ell_{val}(y_i, \mathcal{S}(x_i, \Theta, p)) \tag{9}$$

where (x_i, y_i) indicates the input and corresponding label, ℓ_{train} and ℓ_{val} are the cross-entropy classification loss of the networks, $\mathbb{B}_t$ denotes the sampled batches from $\mathbb{D}_{train}$, and so is $\mathbb{B}_v$ from $\mathbb{D}_{valid}$. After obtaining the optimized Θ and p, we update these parameters back into the target network $\mathcal{T}$. Naturally, in the next search procedure, the sampling of compact subnetworks can benefit from the experience of previous search. In other words, learned knowledge in the NAS space can be iteratively fed back to the sampler, which further improves search efficiency and training performance. In summary, the complete procedure of our joint Search-and-Training Approach is referred to **Algorithm 2**.

4 Experimental Results

4.1 Experimental Setting

Datasets. We validate our network cropping method on several mainstream datasets. Among them, CIFAR-10 contains 60,000 images categorized into 10 classes. The training set has 5000 images per class, 50,000 images in total. The test set contains 1000 images per class, 10,000 images in total. CIFAR-100 is similar to CIFAR-10. It contains 50,000 training and 10,000 test images, categorized into 100 classes. All images are 32x32 colored ones. ImageNet is a large-scale image classification dataset, containing 1000 classes, 1.28 million training images and 50,000 validation images.

Searching and Training Setup. For searching and training, we randomly extract 80% of the official training images as the training set $\mathbb{D}_{train}$ in **Algorithm 2**, and the rest as the validation set $\mathbb{D}_{val}$. In our implementation, we search the compact networks with thresholds in the range [0.6,0.65,0.7,0.75,0.8] for each layer.The hyper-parameter λ is set to 0.1, γ is set to 2.0 and all parameters and weights are initialized by kaiming normal in Pytorch. For different dataset, we apply different settings: 1) On CIFAR dataset, we use SGD with the momentum of 0.9 and the weight decay of 0.00005 as optimizer. At the beginning, we train the target network coarsely for 100 epochs with batch size 128. The learning rate is started from 0.1 and reduced by cosine scheduler. Then we search $T = 30$ compact network, whose parameters are optimized on $\mathbb{D}_{train}$ for M (M = 40 for $t \leqslant 20$, 30 for $t > 20$) epochs with learning rate of 0.05/0.01, corresponding to different M. And the weights are optimized on $\mathbb{D}_{val}$ for N = 5 epochs with fixed learning rate of 0.001. In the fine-tuning stage, we set batch size of 256, learning rate of 0.01 and optimize the selected compact network until convergence. The number of GPUs we used is consistent with the compared method: for the experiments on CIFAR-10 datasets, we use one NVIDIA Titan XP GPUs for training and searching, and NVIDIA 1080Ti for CIFAR-100. 2) On ImageNet datasets, we optimize the parameters via Adam with weight decay of 0.00001 and the weights via the same SGD as CIFAR. For ResNet model, we coarsely train 40 epochs with an initial learning rate of 0.1 and search $T = 20$ compact networks. M is set to be 10 with learning rate of 0.001 and N is set to be 2. When fine-tuning, we set the initial learning rate to 0.001 and divided by 10 every 20 epochs. All experiments on the ImageNet dataset use 4 NVIDIA Titan XP GPUs with batch size of 256.

On CIFAR-10 Dataset. We evaluate our method on ResNet-20,56,110 and 164. We have compared several recently proposed pruning methods with our method, including the soft filter pruning (SFP) [He *et al.*2018a] method, the method for pruning filters with efficient ConvNet (PFEC) [Li *et al.*2016], the Geometric Median based filter Pruning (FPGM) [He *et al.*2019] method and the Transformable Architecture Search (TAS) [Dong and Yang2019a] method. For fairness, the results of other methods were directly borrowed from their papers. As for the consumed time, we calculate the sum of pre-training time, pruning and tuning time, where pruning time is not reported in their paper, so we obtain it through the code and hyper-parameters provided by [He *et al.*2018a, He *et al.*2019, Dong and Yang2019a]. In **Table 1**, We show comparisons in FLOPs, Accuracy and

Network	Method	FLOPs	Acc/Drop	Pre-trained	Time Pruning	Finetuning
ResNet20	SFP [He *et al.*2018a]	2.43e7/42.2%	90.83/1.37↓	3h49m	1h20m	✗
	FPGM [He *et al.*2019]	2.43e7/42.2%	91.09/1.11↓	3h50m	3h34m	✗
	TAS [Dong and Yang2019a]	2.24e7/45.0%	92.88/0.00↓	✓	7h3m	1h18m
	Ours	2.27e7/45.1%	91.95/0.86↓	✗	1h23m	2h
ResNet56	PFEC [Li *et al.*2016]	9.10e7/27.6%	91.31/1.59↓	-	-	-
	SFP [He *et al.*2018a]	5.94e7/52.6%	93.35/0.24↓	3h2m	1h7m	✗-
	FPGM [He *et al.*2019]	5.94e7/52.6%	93.49/0.32↓	1h41m	8h6m	✗
	TAS [Dong and Yang2019a]	5.95e7/52.7%	93.69/0.77↓	✓	16h56m	2h23m
	Ours	6.32e7/49.7%	93.68/0.73↓	✗	2h20m	3h6m
Resnet110	PFEC [Li *et al.*2016]	1.66e8/38.6%	93.44/0.19↓	-	-	-
	SFP [He *et al.*2018a]	1.50e8/40.8%	93.86/-0.18↓	10h20m	5h20m	✗
	FPGM [He *et al.*2019]	1.21e8/52.3%	93.85/0.17↓	6h3m	4h20m	✗
	TAS [Dong and Yang2019a]	1.19e8/53.0%	94.33/0.64↓	✓	15h11m	5h45m
	Ours	1.08e8/58.0%	94.22/0.61↓	✗	3h50m	3h30m
Resnet164	TAS [Dong and Yang2019a]	1.78e8/28.1%	94.00/1.47↓	✓	47h58m	6h48m
	Ours	1.77e8/28.0%	94.05/1.22↓	✗	6h50m	4h50m

Table 1: Comparison of different pruning algorithms for ResNet on CIFAR10. "Flops/Drop" means the calculation and pruning rate. "Acc/Droped" means accuracy and performance drop. "Pre-trained" represents the time for training the pre training network, "Pruning", represents the time for pruning and "Finetuning" represents the time for the final fine-tuning. Note that the data of SFP and FPGM are from the training log published by the authors, and the data of TAS are from the author's open source code.

consumed Time in CIFAR-10. With fewer or close FLOPs, our method can achieve nearly $2\times$ the speed increase, and even $0.3 \sim 1.0\%$ higher accuracy, compared to [Li *et al.*2016, He *et al.*2018a, He *et al.*2019]. Compared with the best performance method [Dong and Yang2019a], our algorithm performs slightly worse on ResNet20 and 56, has lower accuracy on ResNet110 with lower flops, and has better performance on ResNet164. Although our algorithm is short of accuracy of some models compared with [Dong and Yang2019a], it is $3 \sim 5\times$ faster in speed. Compared with a slight decrease in accuracy, we think lesstraining time is more meaningful to a certain extent. At the same time, you can notice that our baseline is lower than that of [Dong and Yang2019a], which is caused by hardware devices, special training skills andexperienced parameter setting. A more intuitive comparison is "Dropped Acc", on which we arecloser to [Dong and Yang2019a]. Note that only [Dong and Yang2019a] uses the setting of $batchsize = 256$(Set to 128 in other implementations), which means more GPU occupation. We think that searching the depth and width of the network at the same time is the reason for more searching time and higher accuracy.

On CIFAR-100 Dataset. In **Table 2**, we compare the performance of different algorithms on ResNet20, 56, 110 and VGG-16. Because the experimental parameter settings on CIFAR-100 are not provided in [He *et al.*2019, He *et al.*2018a], we do not compare time for the sake of fairness. For pruning the ResNet model, our method can still maintain the efficiency of other methods, and the performance is close to [He *et al.*2018a,He *et al.*2019] but only slightly worse than [Dong and Yang2019a]. This is because on the CIFAR-100 dataset, there are only 600 training pictures of each category, and only 480 pictures per category in $\mathbb{D}_{train}$, which causes our method to encounter severe under-fitting problems while learning the compact model from scratch. For VGGNet, our method still outperform the sparse based structure selection method(SSS) [Huang and Wang2018] and the genetic algorithm based channel pruning method(GCP) [Hu *et al.*2018].

Network	Method	FLOPs/Droped	Acc/Droped	Time
ResNet20	SFP [He *et al.*2018a]	2.43e7/42.2%	64.37/3.25↓	-
	FPGM [He *et al.*2019]	2.43e7/42.2%	66.86/0.86↓	-
	TAS [Dong and Yang2019a]	2.24e7/45.0%	68.90/-0.21↓	8.3h
	Ours(fixed rate)	2.05e7/49.7%	67.28/1.31↓	5.5h
	Ours	2.27e7/45.9%	66.48/2.11↓	5.0h
ResNet56	SFP [He *et al.*2018a]	5.94e7/52.6%	68.79/2.61↓	-
	FPGM [He *et al.*2019]	5.94e7/52.6%	69.66/1.75↓	-
	TAS [Dong and Yang2019a]	6.12e7/51.3%	72.25/0.93↓	20.0h
	Ours(fixed rate)	6.72e7/51.1%	70.07/2.86↓	11.0h
	Ours	6.72e7/51.1%	70.63/2.26↓	11.5h
Resnet110	SFP [He *et al.*2018a]	1.21e8/52.3%	71.28/2.86↓	-
	FPGM [He *et al.*2019]	1.21e8/52.6%	72.55/1.59↓	-
	TAS [Dong and Yang2019a]	1.20e8/51.3%	73.16/1.90↓	34.5h
	Ours(fixed rate)	1.08e8/58.0%	71.69/2.73↓	13.0h
	Ours	1.08e8/58.0%	72.26/2.16↓	9.5h
VGG-16	GCP [Hu *et al.*2018]	3.82e8/39.1%	72.01/1.18↓	-
	SSS [Huang and Wang2018]	3.08e8/50.6%	72.95/0.24↓	-
	Ours	3.22e8/48.4%	74.63/1.12↓	4.5h

Table 2: Comparison of different pruning algorithms for ResNet on CIFAR-100. Where "fixed rate" means pruning with a fixed pruning rate for each layer in the network.

Method	FLOPs/Droped	Acc/Droped Top-1	Acc/Droped Top-5	Time
SFP [He *et al.*2018a]	2.38e9/41.8%	74.61/1.54↓	92.06/0.81↓	130h
FPGM [He *et al.*2019]	2.36e9/42.2%	75.50/0.65↓	92.63/0.21↓	112h
TAS [Dong and Yang2019a]	2.31e9/43.5%	76.20/1.26↓	93.07/0.48↓	170h
Ours	2.25e9/44.9%	75.51/1.01↓	92.43/0.66↓	90h

Table 3: Comparison of different pruning algorithms for ResNet-50 model on ImageNet.

On ImageNet Dataset. In **Table 3**, we use 4 NVIDIA 2080Ti GPUs to validate our method on the ResNet-50 model. Compared to the other state-of-the-art algorithm, we still show competitive results. For example, we can prune ResNet-50 in 88 hours, and the prunned network can achieve $75.51\%/92.43\%$ accuracy. The training time of the complete ResNet-50 model is nearly 80 hours, which means that we can complete the entire search algorithm in the same time as other algorithms training unpruned models.

4.2 Ablation Study

In order to show the effect of flexible pruning, we implemented a baseline version of the proposed method, that is, pruning with a manually determined fixed crop rate (the "fixed rate" method in **Table 2**). We calculate the appropriate pruning rate based on the implementation based on flexible pruning and apply it to each layer of the network (as shown top of **Figure 2**). We verified ResNet-20,56 and 110 on the CIFAR-100 dataset. Compared with a fixed pruning rate, the flexible pruning method can effectively speed up the search and improve the performance of the searched network.For ex-

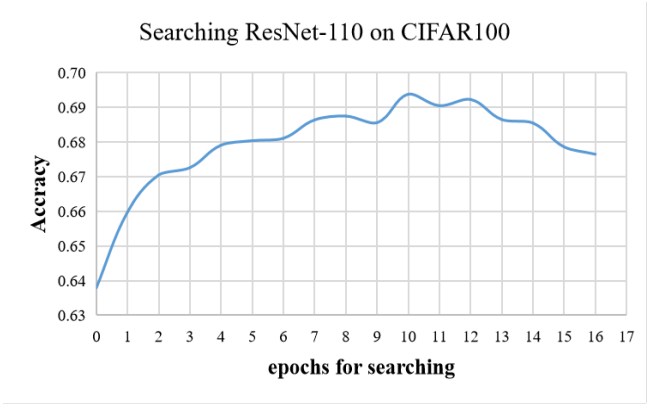

Figure 4: We show the performance of all sampled subnetworks during the search of resnet-110 on Cifar100 dataset. Before the search, the target network has been coarsely trained; after the search, the best performing sub-network will be fine tuned to convergence.

ample, when cropping resnet56, our method based on flexible pruning achieved a 0.57% performance improvement in less time than the fixed cropping method. Note that we are just training a simple fully connected network as a ThresNet with a basic reinforcement learning strategy, so the performance of our algorithm can also be improved by using more powerful networks and more effective reinforcement learning methods.

A very important part of our approach is to select the best performing sub-networks for fine-tuning at the end of the training-search process, so to better illustrate the search-and-training procedure, we provide the performance of all sampled networks. In fact, after the search-and-training procedure (corresponding to lines 4-12 in **Algorithm 2**), we actually get some pretty good but not well-trained sub-networks. Then, as described in **Algorithm 2**, line 13, we select and optimize the network with the best performance. In our method, it is unnecessary to ensure that every sub-network is well-trained after search-and-training procedure. And there are two reasons: first, training every sampled sub-network to convergence will greatly improve the time cost and calculation cost of the algorithm; second, in the process of searching, a well-trained sub-network will lead to the reduction of search space, which limits the sub-network to be searched later. Therefore, the bias towards the last sampled network that you're worried about won't happen either. In fact, in our experiments, the best performing sub-network usually appeared in the middle of the training epochs, as shown in **Figure 4**

5 Conclusion

In this paper, we propose a new joint search and training method designed to learn a compact network directly from scratch. First, we propose to iteratively sample and update compact networks to approximate the target network. Unlike most previous methods, we do not follow the rule of pruning-after-training, but jointly search and train to learn compact networks directly from unclipped networks. Secondly, we have developed a flexible pruning method that flexibly adjusts the weight and size of each layer by learning a set of filter-related weights. For this part, we propose to optimize the ThresNet by reinforcement learning. On some benchmark datasets and network architectures, our proposed method achieves transcendental efficiency improvements while achieving competitive classification accuracy.

Acknowledgements

This work was supported in part by the National Key RD Program of China under Grant 2018AAA0101400 and the Natural Science Foundation of China under Grant 61991451, Grant 61632019, Grant 61621005, and Grant 61836008.

References

[Alvarez and Salzmann, 2016] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[Ashok *et al.*, 2017] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. *arXiv preprint arXiv:1709.06030*, 2017.

[Cai *et al.*, 2018] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[Dai *et al.*, 2019] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11398–11407, 2019.

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

[Dong and Yang, 2019a] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, pages 759–770, 2019.

[Dong and Yang, 2019b] Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3681–3690, 2019.

[Dong and Yang, 2019c] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1761–1770, 2019.

[Frankle and Carbin, 2018] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[Gu *et al.*, 2018] Bin Gu, Miao Xin, Zhouyuan Huo, and Heng Huang. Asynchronous doubly stochastic sparse kernel learning, 2018.

[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 International Joint Conference on Artificial Intelligence*, 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 *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[He *et al.*, 2019] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

[Hu *et al.*, 2018] Yiming Hu, Siyang Sun, Jianquan Li, Xingang Wang, and Qingyi Gu. A novel channel pruning method for deep neural network compression. *arXiv preprint arXiv:1805.11394*, 2018.

[Huang and Wang, 2018] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 304–320, 2018.

[Kim *et al.*, 2019] Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung. Efficient neural network compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12569–12577, 2019.

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

[Lin *et al.*, 2019] Shaohui Lin, Rongrong Ji, Yuchao Li, Cheng Deng, and Xuelong Li. Toward compact convnets via structure-sparsity regularized filter pruning. *IEEE transactions on neural networks and learning systems*, 2019.

[Liu *et al.*, 2018a] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[Liu *et al.*, 2018b] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[Liu *et al.*, 2019] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3296–3305, 2019.

[Pham *et al.*, 2018] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[Real *et al.*, 2019] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[Ye *et al.*, 2018] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.

[You *et al.*, 2019] Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Yingyan Lin, Zhangyang Wang, and Richard G Baraniuk. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

[Zhou *et al.*, 2016] Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pages 662–677. Springer, 2016.

[Zoph *et al.*, 2018] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.