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

Xiaotong Lu¹, Han Huang¹, Weisheng Dong*, ¹, Xin Li², Guangming Shi¹
¹Xidian University
²West Virginia University

{xiatonglu47, 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-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 beliefs are not necessarily true for several reasons [Frankle and Carbin 2018, Liu et al. 2018b, Dettmers and Zettlemoyer 2019, Gu et al. 2018]. First, so-called lottery ticket hypothesis [Frankle and Carbin 2018] 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 Zettlemoyer 2019], 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 Yang 2019a], 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 Yang 2019c]. Af-
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 an $1 \times 1$ 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.
The complete training loss function is given by:

$$\ell_{\text{cross}} = \text{CrossEntropyLoss}(y_i, T(x_i, \Omega, \alpha))$$

where $(x_i, y_i)$ denote the pair of input/output training data, $\alpha$ is the weight associated with each filter, $\Omega$ refers to the parameters in the target network $T$, $\|\cdot\|_2$ refers to the $\ell_2$ norm regularization of the weights and $\lambda$ is the hyper-parameter.

After coarsely training the target network, we continue to iteratively search and obtain a series of compact networks $S^0, S^1, \ldots, S^f$ 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 $\Phi$ - can make a sequence of decisions about the threshold $\beta^l$ according to the filter weight $\alpha^l$. 
where \( \beta^l = \Phi(\alpha^l) \) (2)

The ThresNet takes \( \alpha^l \) 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 \( \alpha^l \). For example, the varying shape sizes of \( \alpha^l \) 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 \( S \) 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.

\[
R = -(l_{\text{avg}} + \gamma \times \text{param}(S))
\]

(3)

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

\[
\nabla_\theta J(\Phi) \approx \frac{1}{K} \sum_{k=1}^{K} \sum_{l=1}^{L} R_k \nabla \Phi \log \pi_\theta (\beta^l|\alpha^l) \quad (4)
\]

where \( R_k \) is the reward computed at the k-th episode and \( \beta^l \) 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 ThresNet will be exploited to search for a compact sub-network \( S \) next.

3.3 Sampler: Search the Compact Subnetwork

Given a target Network \( T \) with \( L \) layers, we denote all convolution filters in the \( l \)-th layer by \( W^l_1, W^l_2, \cdots, W^l_{C_l} \in \mathbb{R}^{C_{l-1} \times k \times k} \), where \( C_l \) 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:

\[
\mathbb{O}^l = \text{Concat}(\mathbb{I}^l * W^l_1, \mathbb{I}^l * W^l_2, \cdots, \mathbb{I}^l * W^l_{C_l})
\]

(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 convolutional filter more directly, we introduce a filter-related weight \( \alpha^l \in \mathbb{R}^{C_l} \) into those filters and rewrite Eq. (5) as:

\[
\mathbb{O}^l = \text{Concat}(\mathbb{I}^l * \tilde{\alpha}^l_1 W^l_1, \mathbb{I}^l * \tilde{\alpha}^l_2 W^l_2, \cdots, \mathbb{I}^l * \tilde{\alpha}^l_{C_l} W^l_{C_l})
\]

\[
\quad \text{s.t.} \quad \tilde{\alpha}^l_c = \frac{\exp(\alpha^l_c)}{\sum_{k=1}^{C_l} \exp(\alpha^l_k)}
\]

(6)

Taking filters and their associated weights \( \alpha^l \) 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 ThresNet 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 \( W^l_1, W^l_2, \cdots, W^l_{C_l} \) for the \( l \)-th layer based on input weights.

3.4 Joint Search-and-Training

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

\[
\mathbb{O}^l = \text{Concat}(\mathbb{I}^l * \hat{p}^l_1 W^l_1, \mathbb{I}^l * \hat{p}^l_2 W^l_2, \cdots, \mathbb{I}^l * \hat{p}^l_{C_l} W^l_{C_l})
\]

\[
\quad \text{s.t.} \quad \hat{p}^l_c = \frac{\exp(p^l_c)}{\sum_{k=1}^{C_l} \exp(p^l_k)}
\]

(7)

where \( p^l \) is the output of Algorithm 1 and \( p^l_k \) indicates the normalized weights for each layer. By sampling each layer of the network, we can get a compact sub-network \( S \) of the

<table>
<thead>
<tr>
<th>Algorithm 1: Sampler</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> ( \alpha^l ) and ( W^l_1, \cdots, W^l_{C_l} )</td>
</tr>
</tbody>
</table>
| Add uniform noise on \( \alpha^l \):
| \( n^l = \alpha^l - \log(\log(u)) \) s.t. \( u \in U(0, 1) \) |
| Calculate the probability of the \( j \)-th candidate:
| \( \hat{n}^l_j = \frac{\exp(n^l_j)}{\sum_{j=1}^{C_l} \exp(n^l_j)} \) |
| Calculate the threshold for \( l \)-th layer’s weigh:
| \( th^l = \text{ThresNet}(\alpha^l) \) |
| for \( c = 1; s < th^l; c++ \) do
| Calculate the sum of the largest \( c \) candidates in \( \hat{n}^l \):
| \( s = \text{sum}(\text{Top}(\hat{n}^l, c)) \) |
| \( p^l = \text{Top}(\alpha^l); c \) |
| \( W^l_1, \cdots, W^l_{C_l} = \text{Top}(W^l_1, \cdots, W^l_{C_l}; c) \) |
| **Output:** \( p^l \) and \( W^l_1, W^l_2, \cdots, W^l_{C_l} \) |
target network $\mathcal{T}$, whose parameters and weights can be expressed by $\Theta$ 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 $\mathbb{D}_{\text{train}}$ and validation sets $\mathbb{D}_{\text{valid}}$ successively, which can be formulated as:

$$\Theta = \arg\min_{\Theta} \sum_{i=1}^{B_t} \ell_{\text{train}}(y_i, S(x, \Theta, p))$$  \hspace{0.5cm} (8)$$

$$p = \arg\min_{p} \sum_{i=1}^{B_v} \ell_{\text{valid}}(y_i, S(x, \Theta, p))$$  \hspace{0.5cm} (9)$$

where $(x, y_i)$ indicates the input and corresponding label, $\ell_{\text{train}}$ and $\ell_{\text{valid}}$ are the cross-entropy classification loss of the networks, $B_t$ denotes the sampled batches from $\mathbb{D}_{\text{train}}$, and so is $B_v$ from $\mathbb{D}_{\text{valid}}$. After obtaining the optimized $\Theta$ 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.

Algorithm 2: Search-and-Training Algorithm

Input: The whole available training set $\mathbb{D}$
1. Initialize the target network $\mathcal{T}$ with filter-related weight $\alpha$ using Eq. (6);
2. Split the whole training set $\mathbb{D}$ into $\mathbb{D}_{\text{train}}$ and $\mathbb{D}_{\text{valid}}$;
3. Coarsely train the target network $\mathcal{T}$ on $\mathbb{D}$ by Eq. (1);
4. for $t = 1 \rightarrow T$ do
   for $m = 1 \rightarrow M$ do
      Sample a sub-network $S'(\Theta, p) \subset \mathcal{T}$ based on Algorithm 1;
      Sample train batch $\mathbb{B}_t = \{(x_i, y_i)\}_{i=1}^{\text{batch}}$ from $\mathbb{D}_{\text{train}}$;
      Optimize $\Theta$ on the $\mathbb{B}_t$ by Eq. (8);
      Sample valid batch $\mathbb{B}_v = \{(x_i, y_i)\}_{i=1}^{\text{batch}}$ from $\mathbb{D}_{\text{valid}}$;
      Optimize $p$ on the $\mathbb{B}_v$ by Eq. (9);
   Update $\mathcal{T}$ via optimized $\Theta$ and $\alpha$;
5. Pick up the best performing network $S$ and fine-tune it on $\mathbb{D}$ until convergence;
Output: A compact network $S$

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 $\mathbb{D}_{\text{train}}$ in Algorithm 2, and the rest as the validation set $\mathbb{D}_{\text{val}}$. 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 $\lambda$ is set to 0.1, $\gamma$ 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 $\mathbb{D}_{\text{train}}$ for $M = 40$ for $t = 20, 30$ for $t > 20$ epochs with learning rate of 0.05/0.01, corresponding to different M. And the weights are optimized on $\mathbb{D}_{\text{val}}$ 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 (PPEC) [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 calculated by [He et al.2018a, He et al.2019, Dong and Yang2019a]. In Table 1, We show comparisons in FLOPs, Accuracy and
consumed Time in CIFAR-10. With fewer or close FLOPs, our method can achieve nearly 2× the speed increase, and even 0.3 ~ 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 less 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 ~ 5× faster in speed. Compared with a slight decrease in accuracy, we think less training 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 and experienced parameter setting. A more intuitive comparison is "Dropped Acc", on which we are closer 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 selecting 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 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].

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.

#### Table 1: Comparison of different pruning algorithms for ResNet on CIFAR10. "Flops/Drop" means the calculation and pruning rate. "Acc/Dropped" 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.

<table>
<thead>
<tr>
<th>Network</th>
<th>Method</th>
<th>FLOPs</th>
<th>Acc/Drop</th>
<th>Pre-trained</th>
<th>Time</th>
<th>Pruning</th>
<th>Finetuning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet20</td>
<td>SFP [He et al. 2018a]</td>
<td>2.4e+09 2.5% 92.80/0.00</td>
<td>24h40m</td>
<td>53h50m</td>
<td>2h</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PFPGM [He et al. 2019]</td>
<td>2.4e+09 2.5% 92.80/0.00</td>
<td>24h40m</td>
<td>53h50m</td>
<td>2h</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TAS [Dong and Yang2019a]</td>
<td>2.4e+09 2.5% 92.80/0.00</td>
<td>24h40m</td>
<td>53h50m</td>
<td>2h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet56</td>
<td>SFP [He et al. 2018a]</td>
<td>5.9e+08 41.4% 72.68/0.00</td>
<td>16h56m</td>
<td>2h20m</td>
<td>3h6m</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PFPGM [He et al. 2019]</td>
<td>5.9e+08 41.4% 72.68/0.00</td>
<td>16h56m</td>
<td>2h20m</td>
<td>3h6m</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TAS [Dong and Yang2019a]</td>
<td>5.9e+08 41.4% 72.68/0.00</td>
<td>16h56m</td>
<td>2h20m</td>
<td>3h6m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet110</td>
<td>SFP [He et al. 2018a]</td>
<td>6.1e+09 43% 72.68/0.00</td>
<td>4.5h</td>
<td>1h23m</td>
<td>2h</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PFPGM [He et al. 2019]</td>
<td>6.1e+09 43% 72.68/0.00</td>
<td>4.5h</td>
<td>1h23m</td>
<td>2h</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TAS [Dong and Yang2019a]</td>
<td>6.1e+09 43% 72.68/0.00</td>
<td>4.5h</td>
<td>1h23m</td>
<td>2h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet164</td>
<td>SFP [He et al. 2018a]</td>
<td>2.2e+10 44% 72.68/0.00</td>
<td>90h</td>
<td>13h</td>
<td>9.5h</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PFPGM [He et al. 2019]</td>
<td>2.2e+10 44% 72.68/0.00</td>
<td>90h</td>
<td>13h</td>
<td>9.5h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Network</th>
<th>Method</th>
<th>FLOPs/Dropped</th>
<th>Acc/Dropped</th>
<th>Top-1</th>
<th>Top-5</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet20</td>
<td>SFP [He et al. 2018a]</td>
<td>2.3e+09 4.1%</td>
<td>74.61/1.54</td>
<td>92.06/0.81</td>
<td>130h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PFPGM [He et al. 2019]</td>
<td>2.3e+09 4.1%</td>
<td>74.61/1.54</td>
<td>92.06/0.81</td>
<td>130h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TAS [Dong and Yang2019a]</td>
<td>2.3e+09 4.1%</td>
<td>74.61/1.54</td>
<td>92.06/0.81</td>
<td>130h</td>
<td></td>
</tr>
<tr>
<td>Ours (fixed rate)</td>
<td>2.2e+09 4.0%</td>
<td>74.61/1.54</td>
<td>92.06/0.81</td>
<td>130h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>FLOPs/Dropped</th>
<th>Acc/Dropped</th>
<th>Top-1</th>
<th>Top-5</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFP [He et al. 2018a]</td>
<td>2.3e+09 4.1%</td>
<td>74.61/1.54</td>
<td>92.06/0.81</td>
<td>130h</td>
<td></td>
</tr>
<tr>
<td>PFPGM [He et al. 2019]</td>
<td>2.3e+09 4.1%</td>
<td>74.61/1.54</td>
<td>92.06/0.81</td>
<td>130h</td>
<td></td>
</tr>
<tr>
<td>TAS [Dong and Yang2019a]</td>
<td>2.3e+09 4.1%</td>
<td>74.61/1.54</td>
<td>92.06/0.81</td>
<td>130h</td>
<td></td>
</tr>
<tr>
<td>Ours (fixed rate)</td>
<td>2.2e+09 4.0%</td>
<td>74.61/1.54</td>
<td>92.06/0.81</td>
<td>130h</td>
<td></td>
</tr>
</tbody>
</table>

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-
ondly, we have developed a flexible pruning method that compact networks directly from unclipped networks. Sec-
like most previous methods, we do not follow the rule of compact networks to approximate the target network. Un-
scratch. First, we propose to iteratively sample and update a method designed to learn a compact network directly from
In this paper, we propose a new joint search and training

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 learn-
ing 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


