Where to Prune

Ding, Guiguang; Zhang, Shuo; Jia, Zizhou; Zhong, Jing; Han, Jungong

Published in:
IEEE Transactions on Image Processing

DOI:
10.1109/TIP.2020.3035028

Publication date:
2020

Citation for published version (APA):
Ding, G., Zhang, S., Jia, Z., Zhong, J., & Han, J. (2020). Where to Prune: Using LSTM to Guide Data-Dependent Soft Pruning. *IEEE Transactions on Image Processing*, *30*, 293 - 304. [9258919]. https://doi.org/10.1109/TIP.2020.3035028

Where to Prune: Using LSTM to Guide Data-dependent Soft Pruning

Guiguang Ding, Shuo Zhang, Zizhou Jia, Jing Zhong, and Jungong Han

***Abstract*—While convolutional neural network (CNN) has achieved overwhelming success in various vision tasks, its heavy computational cost and storage overhead limit the practical use on mobile or embedded devices. Recently, compressing CNN models has attracted considerable attention, where pruning CNN filters, also known as the channel pruning, has generated great research popularity due to its high compression rate. In this paper, a new channel pruning framework is proposed, which can significantly reduce the computational complexity while maintaining sufficient model accuracy. Unlike most existing approaches that seek to-be-pruned filters layer by layer, we argue that choosing appropriate layers for pruning is more crucial, which can result in more complexity reduction but less performance drop. To this end, we utilize a long short-term memory (LSTM) to learn the hierarchical characteristics of a network and generate a global network pruning scheme. On top of it, we propose a data-dependent soft pruning method, dubbed Squeeze-Excitation-Pruning (SEP), which does not physically prune any filters but selectively excludes some kernels involved in calculating forward and backward propagations depending on the pruning scheme. Compared with the hard pruning, our soft pruning can better retain the capacity and knowledge of the baseline model. Experimental results demonstrate that our approach still achieves comparable accuracy even when reducing 70.1% Floating-point operation per second (FLOPs) for VGG and 47.5% for Resnet-56.**[1]

***Index Terms*—Deep learning, model compression, computer vision, image classification**

I. Introduction

In recent years, deep convolutional neural network (CNN) has shown a great success in many computer vision applications, such as image classification [1]–[3], semantic segmentation [4], image captioning [5]–[7], and object detection [8]–[10] and recognition [11]–[13]. However, its success on accuracy has come with a significant amount of model parameters for storage and expensive training costs on GPUs. They all impede the deployment of CNN on computationally limited devices, such as mobile or embedded devices. To remedy this situation, it is desired that CNN can be tiny and fast enough while preserving sufficient accuracy. Therefore, deep model compression has become a popular research topic.

This research was supported by the National Natural Science Foundation of China Grant No. U1936202 and 61925107. (Corresponding author: Jungong Han)

Guiguang Ding, Zizhou Jia and Jing Zhong are with the School of Software, Tsinghua University, Beijing, 100084, China. Email: dinggg@tsinghua.edu.cn.

Shuo Zhang is with Lancaster University, Lancaster, UK.

Jungong Han is with Aberystwyth University, Aberystwyth, SY23 3DB, UK. Email: jungonghan77@gmail.com.

[1]To encourage further developments, the source code is publicly available at https://github.com/Zhong-J/Where-to-Prune

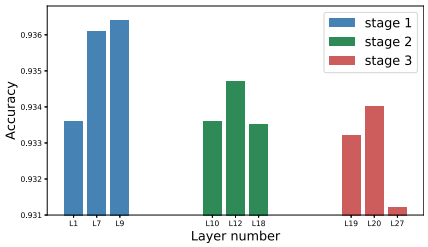

Fig. 1: The accuracy of ResNet-56 after one layer is pruned. The same number of filters are removed in different layers at the same stage. Lm in x-axis denotes the m^{th} residual block.

Current CNN compression techniques can be divided into four categories. The first category makes use of the *quantization* technique, where a model binarization approach [14] is usually adopted. The second category is network *sparseness* [15], [16] by removing unimportant weights or setting it to zero. The third category takes advantage of *tensor factorization* technique [17] which combines small tensors and simple operations to approximate large and complicated tensors. The last category is called *filter-wise pruning* [18], [19] which directly removes unimportant convolutional filters. The filter pruning method preserves the structure of the network, thus resulting in smaller and faster models with little or no performance drop. Therefore, the filter pruning technique has gained the most attention in recent years, which is the focus of this paper.

Basically, deep CNN model compression is a systematic task and the pruning decision should be a trade-off of all factors of the entire model. However, it seems that most existing pruning approaches [18], [20]–[22] only consider the information of individual filters but fail to take the correlation among layers and filters into consideration. In particular, they mostly focus on evaluating the importance of each filter individually at each layer and the filters are pruned from top to bottom or bottom to top in a layer by layer manner. If it happens to remove a few filters from an important layer, the performance of the overall system may drop significantly. On the contrary, pruning many filters in an unimportant layer may impact little on accuracy but get the model complexity dramatically reduced. To demonstrate this, we take the ResNet-56 as an example. Here, we run the filter pruning experiment three times, called three stages, at each of which we prune filters from three layers respectively. There is no interaction between stages, implying that the experiments at different stages are independent. At each stage, the same amount of filters are removed from each of the three layers so that the

model complexities after pruning are identical. To be specific, we randomly remove the same amount of filters from the layer 10, 12 and 18 when carrying our the stage two experiment, and we show the accuracy of the model after pruning at different layers (e.g., L10, L12, or L18 at stage 2) in Figure 1, on which it is clear that the importance of each layer varies, as the performance difference in accuracy is quite noticeable. Therefore, **where to prune** is a critical issue for pruning and choosing to prune the layers that have smaller impacts on the classification accuracy will lead to more complexity reduction with less performance drop. In Table III, we compare a traditional pruning method [18] with the proposed approach that allows to select to-be-pruned layers. It turns out that our approach achieves a larger pruning rate with sufficient accuracy preserved.

The above phenomenon motivates us to investigate the problem of choosing the appropriate layers to prune. In this paper, we propose a novel approach to evaluate the importance of each layer and choose less important layers to prune. Specifically, considering that the CNN usually exploits a hierarchical structure that can be represented as a string, we employ long short-term memory (LSTM) [23] as an evaluation metric to generate the pruning decision for each layer. The entire algorithm is carried out in two steps: unimportant *layers* finding and unimportant *filters* finding. To detect the unimportant layers in a deep CNN, the LSTM is trained using reinforcement learning with model performance and complexity considered in the reward function. In the subsequent unimportant filters finding step, a channel-based method is adopted, which goes over each filter of a certain neural network layer and finally, a number of filters with the least importance are pruned. Through several pruning iterations, the slimmer model is generated, which preserves the performance of deep CNN while significantly reducing the complexity of the model. In addition, the new slimmer network usually ends up having a compact structure, because the training process of LSTM aids in generating a more efficient architecture.

Existing pruning methods usually adopt a hard pruning strategy to remove the unimportant filters from the network directly, in which the importance evaluation of the filter is often inaccurate due to ambiguous calculations. Specifically, when the filter is unimportant for most image data, it is decided to be an unimportant filter, even if it plays a relatively important role for a small portion of image data. In this case, feature extraction and classification performance on those few images are degraded, which means that employing a hard pruning strategy will definitely give rise to the decrease of the overall accuracy. On the contrary, a soft pruning strategy allows the pruned filters in the previous epoch to be updated in the next epoch during the training procedure. In this way, no filters are physically removed and the model capacity can be recovered from the pruned model.

In this paper, we propose the Squeeze-Excitation-Pruning (SEP), which is a *soft* pruning method. Referring to the slimmer architecture generated by LSTM, we use the SEP module to rebuild the baseline network. Meanwhile, the SEP module is used to generate the importance scores of all filters for each given image. In our soft pruning, all filters of the baseline network are preserved, but for each input image, only a portion of important ones are involved in the forward and backward calculations. That is, no filters are removed from the network. When it comes to different image data, different filters might be *softly* pruned depending on the selection results of SEP. This *data-dependent* soft pruning method retains the capacity and knowledge of the baseline model, thus ensuring better performance. Specifically, the complete framework is shown in Figure 2, and our major contributions are summarized as follows:

- We argue that where to prune is actually a critical issue for CNN model compression, which has long been unfortunately neglected. To this end, propose an end-to-end framework to prune networks in the correct order. Concretely, considering the hierarchical structure of CNN, we employ LSTM as an evaluation model to find the least important layers and thus generate the pruning decision for a given network. LSTM is updated using the policy gradient method with both model performance and complexity as the reward.
- Rather than adopting a hard pruning strategy, we propose the SEP pruning method. SEP is a *data-dependent* soft pruning method, which preserves all the filter parameters, but for each image, only some important ones participate in calculating forward and backward propagations. When the given image is changed, different filters may be softly pruned according to the SEP selection.
- Comprehensive experiments are carried out on several benchmark datasets. The results show that our pruning method is capable of compressing a variety of network structures with comparable accuracy and works well on both convolutional and fully-connected networks. It also reveals that our method learns the sensitivity of each network layer.

II. Related Work

Our work is in relation to weight pruning, filter pruning, compact network design, and neural architecture search, each being elaborated below.

A. Weight Pruning

Removing network connections is an intuitive model compression method, which mostly focuses on evaluating and selecting unimportant connections. [16] assesses the network connections through the second-order derivative information, but it leads to high computational complexity. [24] carries out model compression in three steps: removing connections with the smallest absolute weights values, quantization, and Huffman encoding. [25], [26] regularize neural network parameters by group Lasso penalty, resulting in group-level sparsity. [27] prunes connections between the input and output feature maps with a newly proposed class of parameters called Synaptic Strength.

B. Filter Pruning

Existing software and hardware libraries usually struggle to accelerate weight pruning methods. To address this problem,

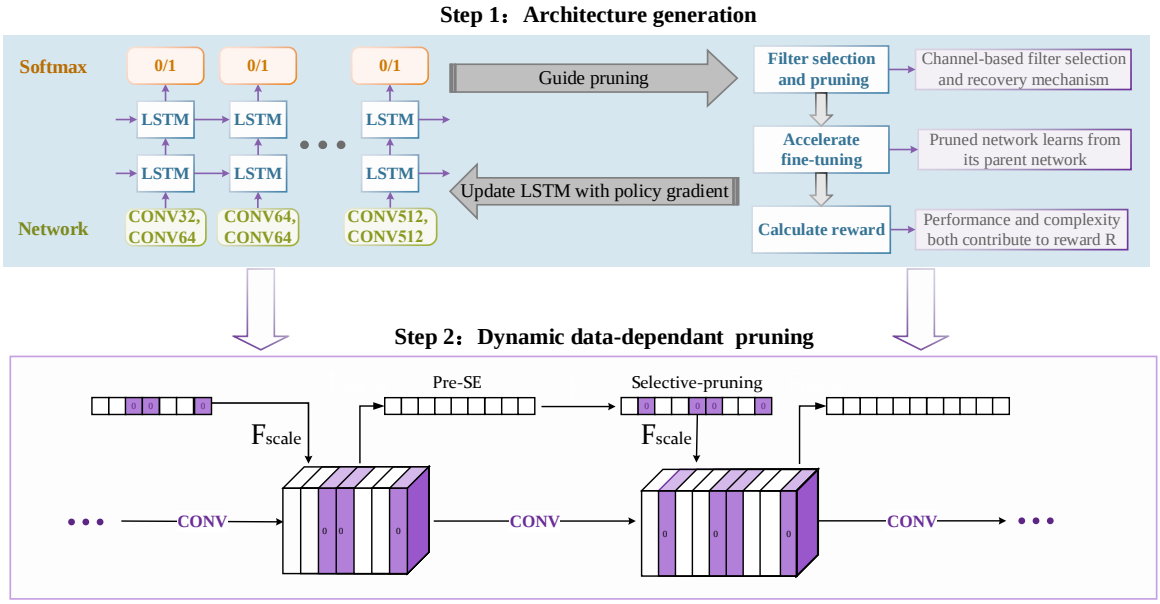

Fig. 2: The framework of our end-to-end pruning method. The first step is to make pruning decisions based on the LSTM evaluation model. After several epochs, a more efficient and slimmer network structure is finally generated. LSTM is updated in the policy gradient method with both model performance and complexity as the reward. In the second step, we rebuild the baseline network by deploying the SEP module for each layer and train it from scratch. The SEP attention module is composed of feature extraction and selection, which generates the weight vector, selects and sets some weights to zero according to the pruning structure generated in the first step. Then, the feature map in the next convolution layer is scaled by this weight vector to achieve dynamic and data-dependent soft pruning.

more works focus on filter-wise pruning. [18] first proposes to prune parameters at the filter level, which evaluates a filter by calculating its absolute weights sum and removes unimportant filters sequentially. [28] assesses the channels by introducing additional discrimination-aware losses to increase the discriminative power of the intermediate layers. ThiNet [21] considers filter pruning to be an optimization problem and proposes a greedy method to prune filters using their statistics information in the next layer. [22] leverages the scaling factors in the batchnorm layers to evaluate filters combined with a sparsity regularization. Different from the methods above, [29] allows the pruned filters to be updated during the training procedure. [30] trains the reinforcement learning agent to predict actions like whether to remove one layer in network and uses a reward function to update their agent. [31] applies the structure regularization to the corresponding out-channels and in-channels in the continuous network layer. [32] utilized reinforcement learning to learn two seperate policies to remove and shrink layers respectively in tiny datasets. [33] uses attention modules as the criterion to evaluate the importance of channels and prune channels with low correlations to accelerate networks. [34] uses reinforcement learning to predict the action and gets sparsity to find the redundancy in each layer for pruning.

C. Compact Network Design

Current widely used compact network structures are designed manually based on expert knowledge. For instance, ShuffleNet [35] is designed specifically for mobile devices with limited computing power, which utilizes two new operations, pointwise group convolution and channel shuffle. As an extension of ShuffleNet, ShuffleNet V2 [36] performs faster and more accurately when following some practical guidelines. Alternatively, MobileNetV1 [37] is based on a streamlined architecture that uses depth-wise separable convolutions to build lightweight deep neural networks. On top of MobileNetV1, MobileNetV2 [38] adds a novel layer module: the inverted residual with a linear bottleneck, which improves the accuracy significantly.

D. Neural Architecture Search

In order to reduce labor costs, automatic design/search of the network structure by machine has received worldwide attention in both academia and industry [39]–[43]. For instance, evolutionary techniques [39]–[41] discover target models from trivial initial architectures by setting up the vast search space, which requires enormous computing resources. Alternatively, [42], [43] utilize the reinforcement learning mechanism to train the recurrent neural network (RNN) controller to generate the neural networks automatically, which have achieved good results on various datasets. To reduce the search space, Nasnet [44] searches for an architectural building block on a small dataset and then transfers it to a larger dataset. For further improvement, Mnasnet [45] introduces a novel factorized hierarchical search space that partitions CNN layers into groups, within which operation and connection searches can be carried out. [46] utilizes Bayesian Optimization to

search for a compressed network structure on a given teacher network.

III. METHOD

In this section, we present our end-to-end pruning method. Before going into further details, we briefly explain the basic idea. We first use LSTM to generate the pruning decisions by evaluating the importance of each layer, where the most unimportant layers will be selected to be pruned. Once we have collected such guidance information, the SEP attention mechanism is employed to rebuild the baseline network. Basically, we train a SEP from scratch by deploying the pruning information, e.g., which layers will be pruned and how many filters in each layer need to be pruned, estimated by the preceding LSTM. The SEP module consists of two parts: the pre-SE module includes squeeze and excitation used for feature extraction and weight vector generation; the selective-pruning module selects and sets some weights to zero. Therefore, the SEP module can automatically predict the importance of each feature map and set some weights to zero based on the information of feature extraction. The details of the end-to-end pruning framework in Figure 2 are elaborated as follows.

(a). **Pruning guidance:** An initial or intermediate network representation is fed into LSTM, and LSTM generates a strategy indicating which layers should be pruned.
(b). **Filter selection and fine-tuning:** We evaluate the importance of each filter in the layers chosen by LSTM with a channel-based method, then prune those unimportant filters combined with the recovery mechanism. Afterwards, we fine-tune the pruned model using the distillation method.
(c). **Updating LSTM:** We update LSTM in a reinforcement learning way incorporating both the performance and complexity of the pruned model in the reward signal.
(d). **Repeat from (a) to (c):**
(e). **Data-dependent soft pruning:** Referring to the slimmer architecture generated by LSTM, the baseline network is rebuilt with SEP modules and trained from scratch to achieve data-dependent soft pruning.

A. Where to Prune

The basic idea can be interpreted as follows: LSTM generates the pruning probability for each layer. The output of each layer is associated with two real values, indicating the probabilities of "Pruning" and "Not Pruning", respectively. Suppose the number of all candidate convolution layers is defined as L, we can get one matrix $P \in \mathbb{R}^{L\times 2}$, corresponding to the pruning probabilities of L conv layers. If the probability of "Not Pruning" is larger than that of "Pruning" in one row, we treat this row as "0", meaning we do nothing for this layer. Otherwise, it is a to-be-pruned layer. In this way, we can obtain a map, indicating which layers in the network need to be pruned.

1) Input and Output of LSTM: A neural network is a hierarchical sequence from input to output connected by operation nodes, which can be convolution, pooling, and fully-connected operation. For a common CNN here, the i^{th} node ξ_i is denoted as (m_i, n_i), where the operation type m is in $\{0, 1, 2\}$ corresponding to convolution, pooling, and fully-connected block respectively, operation attribute n equals to filter number, pooling stride or unit number. Convolution and fully-connected nodes (final classifier layer is not included) are called the primary nodes, while pooling and the final classifier are seen as the secondary nodes because they cannot be pruned but supply auxiliary information instead.

Since LSTM is good at time series prediction, we use a 2-layer LSTM in Figure 2 to learn the network structure and produce reasonable pruning decisions. At each timestep, the current primary node as well as its next primary or secondary node $[\xi_i, \xi_{i+1}]$ are fed into LSTM equivalent to $[m_i, n_i, m_{i+1}, n_{i+1}]$ and the pruning decision whether to prune the first primary node is generated by a softmax layer. For a network with N primary nodes, LSTM repeats the above step N times and N distinct softmax layers predict whether to prune these nodes or not. Pooling nodes and the final classifier, taken as the secondary nodes, cannot get pruning predicted but play a key role in helping LSTM to understand a complete network structure.

2) Training LSTM with Policy Gradient Method: After LSTM generates pruning decisions, we prune some filters in the chosen layers such that a slimmer model can be obtained. Both performance and complexity of this new model contribute to the reward signal R for assessing the performance of LSTM. The trade-off is shown in Eq. 1, where we use the training loss or accuracy on the validation set to measure the $performance$, and use model FLOPs or the number of PARAM to measure the $complexity$. Let λ be a trade-off hyperparameter, whose optimal value can be obtained empirically via experiments:

$$R = performance - \lambda \times complexity. \tag{1}$$

We use the policy gradient algorithm [47] (Eq. 2) here, enabling LSTM to generate better pruning strategies. Concretely, we define α_t, s_t and R_k as Action, State and Reward at time step t of one trajectory respectively. m is the number of rollouts for a single gradient update. In order to reduce the variance cuased by sampled trajectories, the reward of the current input network is taken as our baseline b:

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_\theta \log P(\alpha_t|s_t;\theta)(R_k - b). \tag{2}$$

B. Filter Selection and Fine-tuning Strategy

1) Filter Selection and Recovery: LSTM generates a decision about which layers should be pruned, given an input network. A convolution node in layer i can be denoted by a triplet $\langle \mathcal{I}_i, \mathcal{W}_i, \mathcal{O}_i \rangle$, where $\mathcal{I}_i \in \mathbb{R}^{x_{i-1}\times h_i \times w_i}$ same as $\mathcal{O}_{i-1}$ is the input tensor with channels x_{i-1}, height h_i and width w_i. The filter tensor $\mathcal{W}_i \in \mathbb{R}^{x_i \times x_{i-1}\times k\times k}$ with $k \times k$ filter size convolutes with $\mathcal{I}_i$ and generates an output tensor $\mathcal{O}_i$

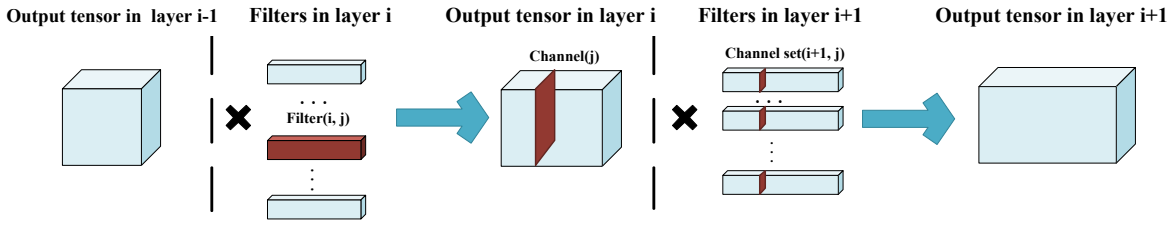

Fig. 3: Pruning a convolution filter requires removing its corresponding convolution channel set in next layer.

with x_i channels. From the perspective of filters, $\mathcal{W}_i$ consists of x_i filters $\mathcal{F}_i \in \mathbb{R}^{x_{i-1}\times k\times k}$, while from the perspective of channels, $\mathcal{W}_i$ consists of x_{i-1} channel sets $\mathcal{C}_i \in \mathbb{R}^{x_i\times k\times k}$.

After the j^{th} filter in the layer i $\mathcal{F}_{i,j}$ has been pruned, its corresponding j^{th} channel set $\mathcal{C}_{i+1,j}$ becomes useless and should be removed at the same time. Convolution structures in other layers are not affected and remain unchanged as shown in Figure 3. It is the output tensor deviation in layer $i+1$ that transfers errors to the final loss and directly leads to worse performance. Therefore we remove less important filters in layer i and channel sets in layer $i+1$ to minimize the output value deviation $\Delta\mathcal{O}_{i+1}$. Since there often exists an activation, pooling or batchnorm layer between two convolution layers, the channel sets $\mathcal{C}_{i+1}$ affect the output value $\mathcal{O}_{i+1}$ more directly than convolution filters $\mathcal{F}_i$. We follow Eq. 3 to measure the importance of each channel set in layer $i+1$ by L2-norm, because L2-norm gives an expectation of the magnitude of the output feature map and reflects the weight diversity. Then, $(R_{prune} \times x_i)$ channel sets with the smallest scores s_j and their corresponding filters in layer i are selected to be removed.

$$s_j = \|(\mathcal{C}_{i+1,j})\|_2, \ s.t.\ j \in [1, x_i]. \tag{3}$$

Eq. 3 is similar to [18] in form, but focuses more on the least important channel sets in layer $i+1$ rather than the least important filters in layer i, which is a reverse selection process starting from the occurrence of loss.

After pruning a channel set and its corresponding filter, $\mathcal{F}_{i+1}$ becomes $\widehat{\mathcal{F}}_{i+1}$ and $\mathcal{O}_{i+1}$ becomes $\widehat{\mathcal{O}}_{i+1}$. To reduce the loss of model performance, we try to minimize $\Delta\mathcal{O}_{i+1}$ through a recovery method. In detail, we use the hyperparameter α to select filters with larger value deviations and scale up those filter tensors by a certain proportion (Eq. 4). Filters here are pruned one by one. After one filter and its corresponding channel set are removed, the recovery mechanism is operated immediately by:

$$\widehat{\mathcal{F}}_{i+1,j} = \begin{cases} \widehat{\mathcal{F}}_{i+1,j} \times (\frac{\|\mathcal{F}_{i+1,j}\|_2}{\|\widehat{\mathcal{F}}_{i+1,j}\|_2})^2, & \text{if } 1 - \frac{\|\widehat{\mathcal{F}}_{i+1,j}\|_2}{\|\mathcal{F}_{i+1,j}\|_2} > \frac{a}{x_i} \\ \widehat{\mathcal{F}}_{i+1,j}, & \text{otherwise} \end{cases}. \\ s.t.\ j \in [1, x_{i+1}] \tag{4}$$

2) Accelerated Fine-tuning: In the LSTM training process, there are many intermediate models produced, then they are fine-tuned to calculate reward signals and update LSTM. In order to improve the algorithm efficiency, we use the distillation method [48] to accelerate the fine-tuning procedure. Specifically, the input model of LSTM is regarded as a teacher network, and the pruned model based on the teacher is taken as a student network. During fine-tuning, we use the loss function g (Eq. 5) to make student's probabilities logit f approximate to teacher's logit z.

$$g(x, z, \theta) = \sum_x \| f(x, \theta), z \|_2^2 . \tag{5}$$

C. Data-dependent Soft Pruning

1) SEP attention module: The slimmer network architecture with the highest reward can be found from the intermediate models generated by LSTM, which becomes the reference of the baseline network rebuilding. The key to the baseline network rebuilding lies in the Squeeze-Excitation-Pruning (SEP) attention mechanism in Figure 4, on which the left part is a normal CNN structure with a few convolution layers while the right part is a SEP module. The SEP module consists of two parts: the pre-SE module similar to SENet [49] includes squeeze and excitation used for feature extraction and weight vector generation, the selective-pruning module selects and sets some weights to zero. We apply the SEP operation to the previous conv layer i, then it generates a weight vector to scale the feature map $\mathcal{O}_{i+1}$ in layer $i+1$. There are two differences between SENet [49] and our pre-SE module. Firstly, we perform the SEP operation on the previous conv layer to predict the weight vector in the next conv layer. Secondly, after squeezing and dimensionality-reduction to x_i/r with the reduction ratio r, the dimensionality is increased to x_{i+1} for the sake of keeping consistency with the dimensionality of $\mathcal{O}_{i+1}$.

We denote the output of the sigmoid function as $\mathcal{V}_{i+1}$, which is the weight vector in the layer $i+1$, and denote the number of filters to prune as m_{i+1}. The slimmer architecture generated by LSTM determines the value of m_i. If the slimmer module has pruned 36 kernels in the third layer, we set m_3 to 36. Regarding the meaning of the attention mechanism, the larger the weight, the more important it will be. We set some weights with minimum values to zero in Eq. 6, because the corresponding feature maps of these weights are of the lowest importance. In Eq. 6, the function F_s is a way of using a sorting method to find the m_{i+1}-th minimum weight in $\mathcal{V}_{i+1}$. For instance, if the slimmer module decided to prune m_{i+1} kernels in the layer $i+1$, we sort the values of $\mathcal{V}_{i+1}$ in ascending order. Then, we get the m_{i+1}-th smallest value represented by $F_s(m_{i+1})$. The activation function $ReLU$ sets all weights smaller than $F_s(m_{i+1})$ to zero. After the activation

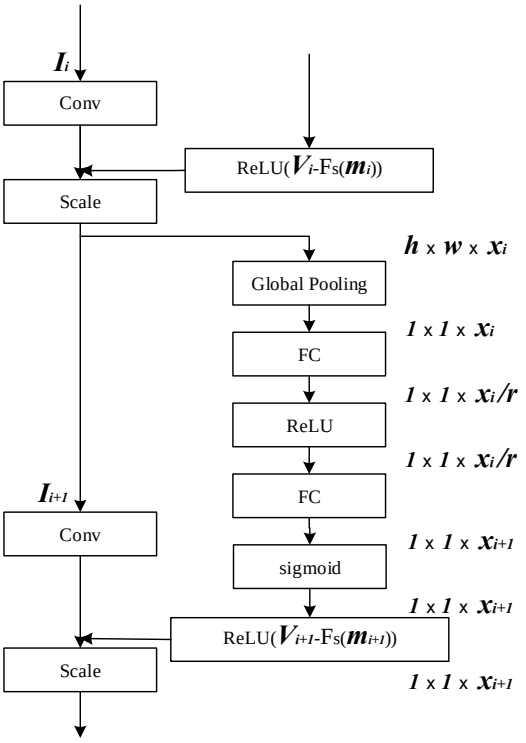

Fig. 4: The schema of the SEP attention module

operation, we get the sparser vector $\widehat{\mathcal{V}}_{i+1}$, which would scale the feature map x_{i+1}.

$$\widehat{\mathcal{V}}_{i+1} = ReLU(\mathcal{V}_{i+1} - F_s(m_{i+1})). \tag{6}$$

2) Architecture Rebuilding: Given the slimmer network architecture generated from LSTM, we deploy the SEP modules on the baseline network to rebuild it. At the core of the SEP module is an attention mechanism, which automatically predicts the importance of each feature map. When the SEP sets some feature maps to zero, it is equivalent to pruning their corresponding kernels. Therefore, when it comes to different image data, different kernels might be utilized according to the SEP selection. All of the kernel parameters are preserved, but only some of which participate in calculating the forward and backward propagations. During the model training, SEP selection strategy is constantly updated, thus indicating this is a dynamic and data-dependent soft pruning procedure. We do not prune the kernels physically, but we preserve all the feature knowledge hierarchy and predict which kernels would be utilized for a specific image data. Although the SEP module introduces extra model complexities, the FLOPs of SEP is negligible because of the fully-connected operation.

In the training process of LSTM, the slimmer model with smaller size and complexity is generated, which can be compared with other hard pruning methods. The SEP algorithm, which is a soft pruning method on the basis of the slimmer architecture generated by LSTM, can also be applied independently, given a predefined slimmer architecture.

D. Discussion

The new model compression method presented above is built upon our previous work in [50], but it differs in many ways from that work, where the main extensions are summarized as: i) We propose a Squeeze-Excitation-Pruning (SEP), which is a data-dependent soft pruning method, retaining the capacity and knowledge of the baseline model. The baseline network is reconstructed with SEP modules and trained from scratch to eventually achieve a dynamic soft pruning; ii) Extensive experiments of SEP method are conducted to validate its performance. Filter distribution maps based on SEP soft pruning are also provided; iii) The experimental results demonstrate that our new proposed SEP method outperforms the previous work [50] and other state-of-the-art pruning methods.

We have to point out that training the LSTM to approximate the slimmer architectures usually takes a long time, which might be a bottleneck of our algorithm. Concretely, it requires several epochs to train each student model before the reward gets maximized. The computational cost of this step can be high, given a large-scale dataset such as ImageNet.

IV. Experiments

We evaluate our method on four benchmark datasets: MNIST [51], CIFAR-10, CIFAR-100 [52], and ImageNet [53]. Two CIFAR datasets contain 50000 training images and 10000 test images. The MNIST contains 60000 and 10000 images for training and testing respectively. In all the datasets, 10% of the images are split from the training set as a validation set used for evaluating new network structures and calculating their reward signals to LSTM. On CIFAR, all images are cropped randomly into 32*32 with four paddings during the training process. Horizontal flip is also adopted. On MNIST, there is no data augmentation preprocessing. In ImageNet, there are over 1.28 million training images and 50K validation images of 1,000 classes.

Three networks: VGGNet [1], ResNet [2] and a 3-layer fully-connected network in [26] are used to validate our method. We employ a 2-layer LSTM with 100 hidden units to make pruning decisions. All the experiments are implemented with PyTorch on one NVIDIA TITAN X GPU.

A. Implementation Details

We train initial models from scratch and calculate their accuracies as baselines. In the first step of training the LSTM, the pruning rate R_{prune} is set to 0.2, and the teacher model instructs the student model to fine-tune 30 epochs on CIFAR and 10 epochs on MNIST dataset. LSTM training is terminated when LSTM no longer produces a better network structure within 10 epochs. We retrain the network with the best reward for 250 epochs on CIFAR and 100 epochs on MNIST. For ImageNet, the pruning rate R_{prune} is set to 0.1, and the teacher model instructs the student model to fine-tune 20 epochs. LSTM training is terminated when LSTM no longer produces a better network structure within 10 epochs. We then retrain the network with the best reward for 40 epochs. Both training

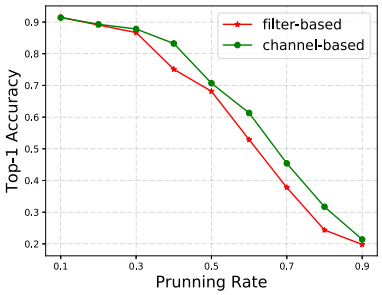

Fig. 5: Comparison between two methods.

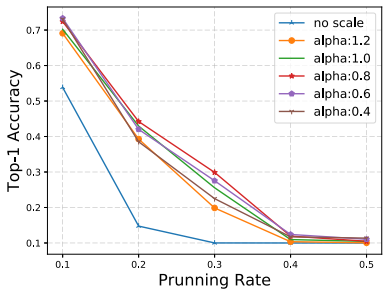

Fig. 6: The effect of recovery mechanism.

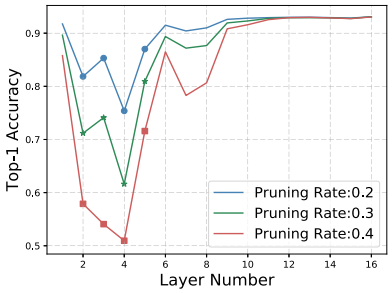

Fig. 7: Sensitivity of VGG-19 for layers.

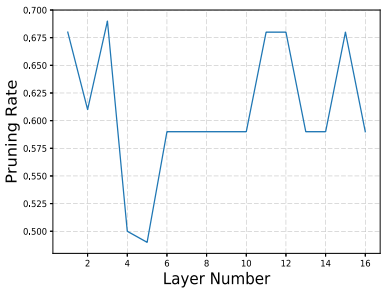

Fig. 8: Pruning rate within 50 epochs.

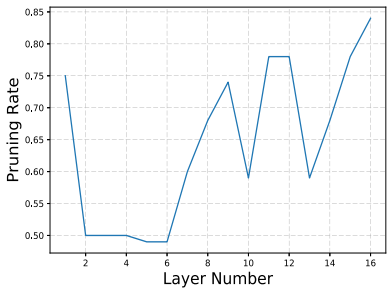

Fig. 9: Pruning rate within 150 epochs.

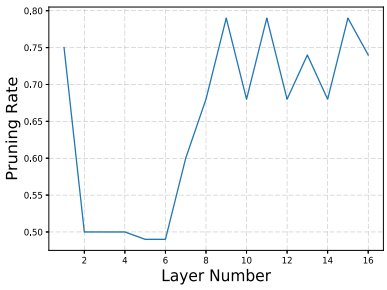

Fig. 10: Pruning rate within 250 epochs.

and validation datasets are used for retraining the network with the fixed learning rate of 0.001 for ultimate accuracy.

The new parent structure is created based on the combination of the current parent structure and the pruning information for each layer generated by LSTM. Then it is added into the list of parent structures for the next training epoch. In each epoch of LSTM training, 5 parent structures with the largest rewards in the list are picked up and fed into the LSTM successively for the next training epoch. Their rewards are taken as baselines b in policy gradient method. If there are no more than 5 local structures, all local networks are taken as inputs. In the first epoch, the input is the pre-trained network. We use FLOPs to measure the complexity of CNN and PARAM to measure fully-connected networks in order to keep in line with the existing methods.

After getting the slimmer model from LSTM, we rebuild the baseline network to deploy the SEP attention modules. The network redeployed is trained from scratch for 90 epochs on ImageNet and 200 epochs on the rest of datasets to get its final accuracy.

TABLE I: Results of VGG-19 on CIFAR-10.

Model	FLOPs	Pruned Rate%	Params	Acc.%
Baseline	3.98×10^8	–	20.04M	93.66
Slimming-4 [a][22]	8.89×10^7	77.2	3.76M	-0.25
Slimming-5 [22]	4.41×10^7	88.7	2.84M	-1.39
Ours-LSTM	5.98×10^7	84.7	2.92M	-0.36
Ours-scratch [b]	5.98×10^7	84.7	2.92M	-1.74
LSTM-SEP	5.98×10^7	84.7	4.01M	-0.26

[a] "Slimming-N" denotes repeating the slimming method [22] N times.
[b] "Ours-scratch" denotes training the slimmer network generated by LSTM from scratch for 200 epochs.

TABLE II: Results of VGG-19 on CIFAR-100.

Model	FLOPs	Pruned Rate%	Params	Acc.%
Baseline	3.98×10^8	–	20.04M	73.26
Slimming-3 [22]	1.27×10^8	67.3	5.48M	-2.34
Slimming-4 [22]	6.63×10^7	83	3.27M	-3.85
Ours-LSTM	1.17×10^8	70.1	4.86M	+0.0
Ours-scratch	1.17×10^8	70.1	4.86M	-3.5
LSTM-SEP	1.17×10^8	70.1	5.95M	+0.31

B. Filter Selection and Recovery Criteria

Our channel-based method is compared with the filter-based method [22], which evaluates a filter by calculating the sum of its absolute weights. We prune some filters from a pre-trained VGG-16 on CIFAR-10 without applying the recovery mechanism. Different layers are pruned with the same pruning rate. Then, we fine-tune the pruned model for 1 epoch. The experiment is repeated 5 times to eliminate the influence of random disturbance, and we report the averaged accuracy on the test set. Figure 5 shows the pruning results with pruning rate, ranging from 0.1 to 0.9, while both methods are set with the same configuration. The results reveal that our channel-based filter selection outperforms the filter-based selection method.

To evaluate the recovery mechanism, we calculate the accuracy of a pruned model on the test set directly without fine-tuning. Figure 6 indicates that the recovery method helps the pruned model to recover its performance significantly. Instead of a random value, we empirically run several experiments to obtain the optimal value for the hyperparameter α. The observation is: while the hyperparameter α in Eq. 4 equals 0.8, the pruned model gets the best recovery. In fact, when we prune a completely unimportant channel, the filter it belongs to

TABLE III: Results of ResNet-56 on CIFAR-10.

Model	FLOPs	Pruned Rate%	Params	Acc.%
Baseline	1.25×10^8	–	0.86M	93.04
PFEC-A [18]	1.12×10^8	10.4	0.77M	+0.06
PFEC-B [18]	9.09×10^7	27.6	0.61M	+0.02
CP [54]	–	50.6	–	-1.00
NISP-56 [55]	–	43.6	–	-0.03
SFP [29]	5.94×10^7	52.6	0.51M	-1.33
FPGM [56]	5.94×10^7	52.6	0.51M	-0.66
Ours-LSTM-1	8.24×10^7	34.1	0.56M	+0.56
Ours-scratch-1	8.24×10^7	34.1	0.56M	-0.85
LSTM-SEP-1	8.24×10^7	34.1	0.62M	+1.01
Ours-LSTM-2	6.56×10^7	47.5	0.49M	-0.11
Ours-scratch-2	6.56×10^7	47.5	0.49M	-0.87
LSTM-SEP-2	6.56×10^7	47.5	0.62M	+0.81

TABLE IV: Results of a Fully-connected Network on MNIST.

Model	Pruned%	Acc.%	#Neurons
Baseline	–	98.57	784-500-300-10
Structured sparsity [26]	83.5	-0.11	434-174-78-10
Slimming-1 [22]	84.4	-0.06	784-100-60-10
Ours-LSTM	87.26	-0.03	784-83-48-10

does not need to be scaled because the pruned channel almost has no contribution to the whole network performance. Unless it covers a relatively large proportion of the filter, scaling operation is unnecessary.

C. Results

Our layer-selection method based on LSTM is compared to both orderly and global hard filter pruning methods. Specifically, on the fully-connected network and VGG, we report the pruning results compared with two global pruning methods [22], [26]. On the Resnet-56, we compare our pruning method with a series of existing pruning methods, including an orderly pruning method [18], CP [54], NISP-56 [55], and FPGM [56]. We also provide SEP results to reveal the performance of data-dependent soft pruning.

1) VGG-19 on CIFAR-10: We prune the VGG-19 [1] on the CIFAR-10 dataset. Each convolution layer is followed by a batch normalization layer [59] and we prune filters from the FC layer, which is the last layer before classification.

FLOPs is used as an indicator of model complexity. One multiply-add here is regarded as a floating-point operation unit. We calculate the reward R according to Eq. 1 where network's accuracy in validation set represents $performance$, FLOPs represents $complexity$ and λ is set to 4×10^{-10}. We summarize the results in Table I comparing our layer-selective method and SEP method with the global slimming method [22], which selects unimportant filters in all the layers first and then prune all of them simultaneously. "Slimming-N" denotes repeating the slimming method N times. After LSTM is trained for 150 epochs, the optimal structure emerges, whose FLOPs is reduced by 84.7% with only 0.36% accuracy decreased.

TABLE V: Results of ResNet-50 on ImageNet.

Model	baseline Top-1 Acc.%	baseline Top-5 Acc.%	Pruned Rate%	Params	Top-1 Acc.%	Top-5 Acc. %
SFP [29]	76.15	92.87	41.8	16.96M	-1.54	-0.81
FPGM [56]	76.15	92.87	42.2	16.96M	-1.12	-0.47
CFP [57]	75.30	92.20	49.6	–	-1.90	-0.80
CP [54]	–	92.20	50.0	–	–	-1.40
GDP [58]	75.13	92.30	51.3	–	-3.24	-1.59
Ours-LSTM	76.12	93.00	43.0	15.96M	-1.12	-0.33
LSTM-SEP	76.12	93.00	43.0	17.18M	-0.90	-0.27

We use SEP modules to rebuild the baseline network referring to the optimal slimmer structure generated by LSTM, which gets a better performance with only 0.26% accuracy declined. We also train the slimmer network from scratch for 200 epochs, which is equivalent to a hard filter pruning model, and compare it with the SEP rebuilding model. The results show that our soft pruning can maintain higher precision than the hard pruning.

It is worth noting that [22] takes one multiply-add as two floating-point operations, so their calculated FLOPs is two times as much as ours. For a fair and clear comparison, we convert their FLOPs such that it can be in line with ours. It can be observed that our pruned model is more accurate than the pruned model generated by [22] (-0.26 vs -1.39) when the FLOPs are comparable (84.7% vs 88.7%).

Moreover, the number of parameters (Params) is an indication of the memory costs for storing a trained deep model, which is a widely used criterion for evaluating model pruning algorithms. Since extra FC layers are added into the proposed SEP modules, additional parameters are inevitably produced. As a result, though our LSTM-SEP obtains the best accuracy, the model parameters are increased a little bit, compared to Ours-LSTM method without SEP modules. However, from Table I, we still can see that the accuracy of Ours-LSTM is over 1% higher than that of Slimming-5 [22] when the numbers of the parameters of these two trimmed models are similar (2.92M vs 2.84M).

For further investigation, we plot the sensitivity of each layer in the pre-trained VGG-19 in Figure 7. Specifically, at each time we prune one layer while keeping the other layers unchanged, then calculate the accuracy. The results depict that the overall sensitivity distribution keeps the same under different pruning rates and the most sensitive four layers are layer 2, 3, 4, 5. Figure 8, Figure 9 and Figure 10 represent the practical pruning rates of each layer for the optimal network after training LSTM for 50, 150 and 250 epochs respectively. With more training, the real pruning rate from layer 2 to 5 becomes lower and the other layers are pruned more, which is consistent with the observation from Figure 7. The results demonstrate that our method could make reasonable pruning decisions and learn network sensitivity effectively.

2) VGG-19 on CIFAR-100: We use the same VGG-19 network to evaluate our method on CIFAR-100. Due to more categories, CIFAR-100 is much more difficult to train than

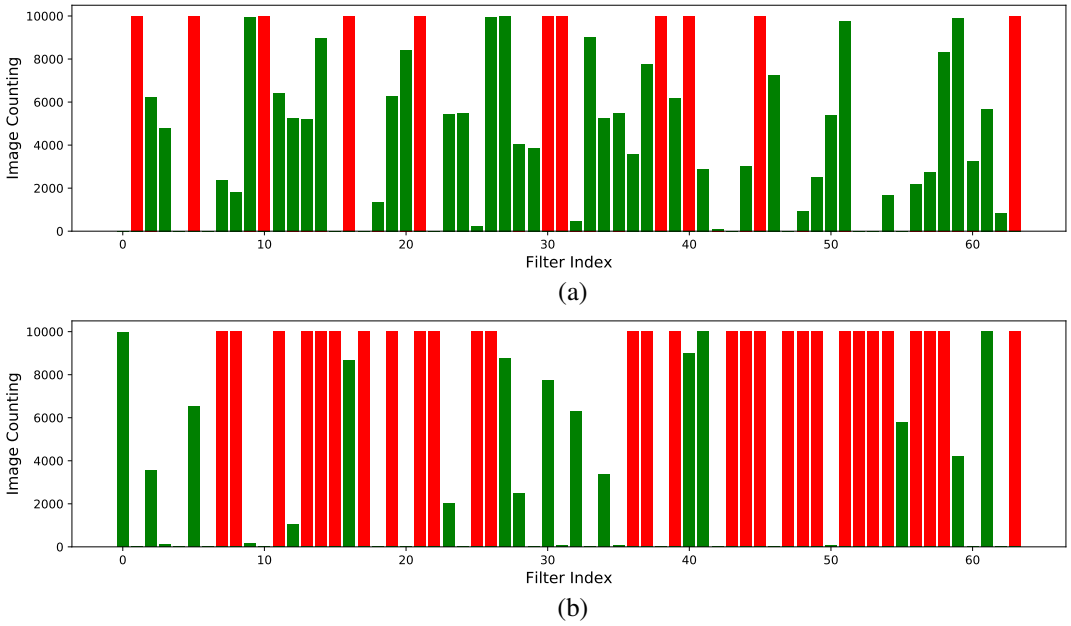

Fig. 11: The soft-pruned filters distribution of the last (a) and the second-to-last (b) residual block in LSTM-SEP-2 model based on ResNet-56 on CIFAR-10 testset. Some filters (red bar) are always discarded on all the images but some (green bar) are discarded on specific images.

CIFAR-10. Thus, the training and validation set are both used to fine-tune the pruned model. Here we use the training loss to evaluate $performance$, and set λ to 2×10^{-11} in Eq. 1. After training LSTM for 123 epochs, we get the best network whose FLOPs is reduced by 70.1% with no accuracy drop. The SEP rebuilding model even improves the accuracy by 0.31%, which indicates the superiority of the SEP module. The attention mechanism improves the model performance. In the meantime, the soft pruning manner retains the capacity of the baseline network, thus maintaining the accuracy to the fullest. As can be seen from Table II, our method outperforms the slimming method significantly (above 3%), even though the number of parameters is a little bit higher than that of Slimming-4 [22].

3) ResNet-56 on CIFAR-10: In this section, we verify the feasibility of our method on ResNet-56 [2]. Due to the particularity of the ResNet structure, we only prune the first convolution layer of each ResNet block and keep the second convolution layer unchanged. The parameter configuration of Eq. 1 is the same as the VGG-19 experiment on CIFAR-10.

Table. III reports the results when compared to [18] [54] [55] [56], which analyze the sensitivity of each ResNet block first, then prune filters referring to the analysis results. Note that all the results of existing algorithms are collected from their original publications. We do not make any analysis in advance because our method is capable of automatically learning the network sensitivity. After LSTM is trained for 32 epochs, the best network emerges with 47.5% FLOPs reduction and comparable accuracy. Compared to [18], more filters are pruned with an acceptable accuracy decrease of 0.11% so that we get the model with the minimum number of parameters, which is only 0.49M. For further comparison, we select the second-best structure with fewer FLOPs reduction, and it achieves a notable 0.56% accuracy promotion. The SEP models based on these two slimmer architectures present more significant performance. Even if compared to a recent method [56], we still obtain the promising results - the accuracy of our best slimmer model exceeds that of their slimmer model by 1.47% when the FLOPs of both models are equivalent.

In Figure.11, we draw the maps of soft-pruned filters distribution in the last and second-to-last residual block respectively to show the data dependence of the SEP method. We run the LSTM-SEP-2 model on CIFAR-10 to check which filters are discarded for different image data in a specific layer. The x-axis represents the filter index, while the y-axis represents the number of images on which this filter is discarded. The last and the second-to-last residual block both consist of 64 filters. The CIFAR-10 testset concludes 10000 images. As can be seen in Figure.11, some filters (red bar) are always discarded on all the images but some (green bar) are discarded on specific images. Some filters are always utilized because they are important to all data. SEP selects different filters for different images, which reveals that the SEP is data-dependent, and has the ability to select different filters for different images.

4) A Fully-connected Network on MNIST: We further validate the effect of our method on multi-layer perceptrons. We prune a 3-layer fully-connected network compared with two global pruning methods [22], [26] as shown in Table IV. Similar to CNN, the evaluation of neurons in the current FC layer depends on its next FC layer. Here we use the accuracy on the validation set to measure $performance$. We set λ to 1×10^{-7}. After 20 epochs, the optimal network structure emerges with 87% neurons pruned and 0.03% accuracy drop.

5) ResNet-50 on Imagenet: In this section, we verify the performance of our method on ImageNet, which is a large-scale dataset. Since ResNet-50 [2] structure is commonly used by many pruning methods such as SFP [29], CP [54],

GDP [58], we choose it to conduct the pruning experiments for a fair comparison. Specifically, we prune the first and second convolution layers of each ResNet block and keep the third convolution layer unchanged. The parameter configuration of Eq. 1 is the same as the VGG-19 and ResNet-56 experiments on CIFAR-10.

Table. V shows that our methods can achieve competitive performance, compared to state-of-the-art methods including SFP [29], FPGM [56], CFP [57], CP [54], GDP [58]. Note that we collect their results from the original publications, and no pre-trained models are used. Seen from the results, the performance of LSTM-SEP is better than that of Ours-LSTM, which shows that our soft pruning can maintain higher accuracy than the hard pruning. Although the pruning rate of our methods is slightly lower than that of CFP [57], CP [54] and GDP [58], the accuracies of Ours-LSTM and LSTM-SEP are much higher than them. Particularly, the accuracy of LSTM-SEP exceeds CFP [57] and GDP [58] models by 1.00% and 2.34% respectively and the Top5 accuracies of our two methods are 1.00% higher than that of CP [54] and GDP [58]. Besides, in comparison to SFP [29] and FPGM [56] that also prune filters in a soft manner, LSTM-SEP reduces more FLOPs of the model with even less accuracy drops. Overall, it is clear that our method outperforms the state-of-the-art soft pruning methods.

V. Conclusion

In this paper, we have presented a framework to evaluate the importance of each network layer and to select the most unimportant layers to prune. Considering the hierarchical structure of CNN, we employ LSTM as an evaluation model to generate pruning decisions. Besides, the channel-based filter selection method and recovery mechanism are adopted to prune filters effectively. Based on the slimmer architecture generated from LSTM, we further propose the SEP attention mechanism to rebuild the baseline network, which realizes the data-dependent soft pruning. Experimental results show the superiority of our methods compared to both orderly and global pruning methods and reveal the ability to learn the sensitivity of each network layer. In future work, we will, on one hand, use LSTM model to guide other filter pruning algorithms, including C-SGD [60] and GSC [61];and on the other hand, apply the proposed model compression method to simplify deep models for different visual tasks, such as video retrieval [62], image classification [63] and 3D reconstruction [64].

Acknowledgment

The authors would like to thank the anonymous reviewers for the constructive suggestions.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[4] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CVPR*, 2016.
[5] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *CVPR*, 2015.
[6] S. Ye, J. Han, and N. Liu, "Attentive linear transformation for image captioning," *IEEE Transactions on Image Processing*, vol. 27, pp. 5514–5524, 2018.
[7] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *ICML*, 2015.
[8] R. Girshick, "Fast r-cnn," in *ICCV*, 2015, pp. 1440–1448.
[9] Y. Zhu, C. Zhao, H. Guo, J. Wang, X. Zhao, and H. Lu, "Attention couplenet: Fully convolutional attention coupling network for object detection," *IEEE Transactions on Image Processing*, pp. 113–126, 2019.
[10] G. Cheng, J. Han, P. Zhou, and D. Xu, "Learning rotation-invariant and fisher discriminative convolutional neural networks for object detection," *IEEE Transactions on Image Processing*, pp. 265–278, 2019.
[11] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015.
[12] X. Zhang, H. Luo, X. Fan, W. Xiang, Y. Sun, Q. Xiao, W. Jiang, C. Zhang, and J. Sun, "Alignedreid: Surpassing human-level performance in person re-identification," *arXiv preprint arXiv:1711.08184*, 2017.
[13] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *CVPR*, 2015, pp. 815–823.
[14] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*. Springer, 2016, pp. 525–542.
[15] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, vol. 2, 1990, pp. 598–605.
[16] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in neural information processing systems*, 1993, pp. 164–171.
[17] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in neural information processing systems*, 2014, pp. 1269–1277.
[18] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *ICLR*, 2017.
[19] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," in *ICLR*, 2017.
[20] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.
[21] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," *arXiv preprint arXiv:1707.06342*, 2017.
[22] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," *arXiv preprint arXiv:1708.06519*, 2017.
[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[24] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR*, 2016.
[25] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini, "Group sparse regularization for deep neural networks," *Neurocomputing*, 2017.
[26] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.
[27] C. Lin, Z. Zhong, W. Wu, and J. Yan, "Synaptic Strength For Convolutional Neural Network," *NIPs*, 2018.
[28] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware Channel Pruning for Deep Neural Networks," *NIPs*, 2018.
[29] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks," in *IJCAI*, 2018.
[30] R. Pahwa, M. G. Arivazhagan, A. Garg, S. Krishnamoorthy, R. Saxena, and S. Choudhary, "Data-driven compression of convolutional neural networks," *arXiv preprint arXiv:1911.12740*, 2019.
[31] J. Li, Q. Qi, J. Wang, C. Ge, Y. Li, Z. Yue, and H. Sun, "Oicsr: Out-in-channel sparsity regularization for compact deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7046–7055.

[32] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani, "N2n learning: Network to network compression via policy gradient reinforcement learning," *arXiv preprint arXiv:1709.06030*, 2017.

[33] K. Yamamoto and K. Maeno, "Pcas: Pruning channels with attention statistics for deep network compression," *arXiv preprint arXiv:1806.05382*, 2018.

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

[35] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," *arXiv preprint arXiv:1707.01083*, 2017.

[36] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design," *ECCV*, 2018.

[37] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[38] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *arXiv preprint arXiv:1801.04381*, 2018.

[39] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[40] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[41] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 497–504.

[42] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.

[43] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *ICLR*, 2016.

[44] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *arXiv preprint arXiv:1707.07012*, 2017.

[45] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," *arXiv preprint arXiv:1807.11626*, 2018.

[46] S. Cao, X. Wang, and K. M. Kitani, "Learnable embedding space for efficient neural architecture compression," *arXiv preprint arXiv:1902.00383*, 2019.

[47] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[48] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Advances in neural information processing systems*, 2014, pp. 2654–2662.

[49] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," *arXiv preprint arXiv:1709.01507*, 2017.

[50] J. Zhong, G. Ding, Y. Guo, J. Han, and B. Wang, "Where to prune: Using lstm to guide end-to-end pruning," *IJCAI*, 2018.

[51] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits, 1998," *URL http://yann. lecun. com/exdb/mnist*, vol. 10, p. 34, 1998.

[52] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Tech Report*, 2009.

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

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

[55] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9194–9203.

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

[57] P. Singh, V. K. Verma, P. Rai, and V. Namboodiri, "Leveraging filter correlations for deep model compression," in *The IEEE Winter Conference on Applications of Computer Vision*, 2020, pp. 835–844.

[58] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang, "Accelerating convolutional networks via global & dynamic filter pruning." in *IJCAI*, 2018, pp. 2425–2432.

[59] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015, pp. 448–456.

[60] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal sgd for pruning very deep convolutional networks with complicated structure," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4943–4953.

[61] X. Ding, G. Ding, X. Zhou, Y. Guo, J. Han, and J. Liu, "Global sparse momentum sgd for pruning very deep neural networks," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 6382–6394.

[62] G. Wu, J. Han, Y. Guo, L. Liu, G. Ding, Q. Ni, and L. Shao, "Unsupervised deep video hashing via balanced code for large-scale video retrieval," *IEEE Transactions on Image Processing*, vol. 28, no. 4, pp. 1993–2007, 2019.

[63] Y. Guo, G. Ding, J. Han, and Y. Gao, "Zero-shot learning with transferred samples," *IEEE Transactions on Image Processing*, vol. 28, no. 4, pp. 1993–2007, 2019.

[64] C. Yan, B. Shao, H. Zhao, R. Ning, Y. Zhang, and F. Wu, "3d room layout estimation from a single rgb image," *IEEE Transactions on Multimedia*, vol. DOI: 10.1109/TMM.2020.2967645.

Guiguang Ding is currently an Associate Professor with the School of Software, Tsinghua University, China. Before joining the School of Software in 2006, he was a Post-Doctoral Research Fellow with the Department of Automation, Tsinghua University. He has published over 100 papers in major journals and conferences, including the IEEE TIP, TMM, TKDE, SIG IR, AAAI, ICML, IJCAI, CVPR, and ICCV. His current research centers on the areas of multimedia information retrieval, computer vision, and machine learning.

Shuo Zhang is currently a Ph. D. candidate with School of Computing and Communications at Lancaster University, Lancaster, UK. Previously, he obtained his M.Sc. degree from the University of Sheffield, Sheffield, UK. His research focuses on very deep Neural Network compression and acceleration.

Zizhou Jia is currently a graduate student with School of software engineering at Tsinghua University, Beijing, China. Previously, he obtained his B.S degree from the Wuhan University, Wuhan, China. His research focuses on deep Neural Network compression and acceleration.

Jing Zhong received her B. S. degree from School of Computer Sicience, Beijing Institute of Technology, China in 2016. And currently she is a M.E. candidate in School of Software in Tsinghua University. Her research interests include computer vision, machine learning and model compression.

Jungong Han is currently a Chair Professor with the Department of Computer Science, Aberystwyth University, U.K. He also holds an honorary professorship with the University of Warwick, U.K. Previously, he was working at Lancaster University. His research interests include computer vision, artificial intelligence, and machine learning.