PRUNING DEPTHWISE SEPARABLE CONVOLUTIONS FOR EXTRA EFFICIENCY GAIN OF LIGHTWEIGHT MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep convolutional neural networks are good at accuracy while bad at efficiency. To improve the inference speed, two kinds of directions are developed, lightweight model designing and network weight pruning. Lightweight models have been proposed to improve the speed with good enough accuracy. It is, however, not trivial if we can further speed up these "compact" models by weight pruning. In this paper, we present a technique to gradually prune the depthwise separable convolution networks, such as MobileNet, for improving the speed of this kind of "dense" network. When pruning depthwise separable convolutions, we need to consider more structural constraints to ensure the speedup of inference. Instead of pruning the model with the desired ratio in one stage, the proposed multi-stage gradual pruning approach can stably prune the filters with a finer pruning ratio. Our method achieves 1.68 times speedup with neglectable accuracy drop for MobileNetV2.

1 INTRODUCTION

Convolution neural networks (CNNs) have become an effective and well-developed technique that achieves state-of-the-art performance on many tasks. However, to gain better accuracy, the architectures of CNNs tend to be larger or deeper, which makes the CNN models infeasible to be realized on resource-limited devices such as home robots and mobile phones.

To overcome this problem, a significant direction in recent studies is to design new lightweight architectures that consume less floating-point operations. Among these researches, depthwise separable convolution (Howard et al. (2017)) becomes a promising design strategy for reducing the model size and complexity of CNNs, where the standard convolution operation of CNN is decomposed into a depthwise convolution layer followed by several 1×1 convolution layers for simplifying the CNN structure. The resulted CNN models such as MobileNetsV1 (Howard et al. (2017)) and MobileNetV2 (Sandler et al. (2018)) have been widely used in real applications. The depthwise separable convolution structure has also become a popular choice in recent neural architecture search (NAS) studies such as Liu et al. (2018), Cai et al. (2019), and Tan et al. (2019).

From another viewpoint, a CNN model often contains much redundancy among their weights after training (Han et al. (2015), Liu et al. (2017)). Hence, a useful strategy to compress CNN models is to remove the superfluous weights or filters of their convolutional kernels (He et al. (2017), Li et al. (2017), Zhu & Gupta (2018), Ding et al. (2019)), so that the model sizes can be reduced and the inference speeds can be increased. Once the unnecessary weights or filters are pruned, the CNN models can be fine-tuned or distilled (Hinton et al. (2015)) for restoring the performance via re-training.

Although the technique of pruning redundant filters has been shown highly effective for compressing standard convolution operations of CNNs, it has not been well applied to depthwise separable convolutions yet. The reasons could be as follows. First, depthwise separable convolution is often performed once per layer. As it is the only operation regarding the spatial context of the input feature map in the layer, reducing it would cause considerable performance degradation. Second, unlike standard convolutional kernels, pruning the depthwise separable convolution involves more sophisticated constraints on both the input and output feature maps in a layer.

To address these issues, we solve the structural constraint over the depthwise separable convolutions. Based on the constraints derived for pruning, we then introduce a method to further remove the redundancy of lightweight structures built up with depthwise separable convolution layers. Our approach is applicable to the tasks where architectures factorization using depthwise separable convolutions (such as MobileNets) for the deployment of embedded devices; this is helpful to earn extra gains of the inference efficiency of CNN models. The characteristics are summarized as follows.

• We derive the constraints required on layer-wised I/O when deleting filters in a channel-wise group convolution, since depthwise separable operations are often realized as group convolutions (Ciresan et al. (2011), Chetlur et al. (2014), Ioannou et al. (2017)).

• A multi-stage gradual pruning approach is introduced to simplify the CNN models and adapt the weights progressively.

2 Related work

In this section, we first give a review on the depthwise-separable structure in Section 2.1. Then, we survey the neural network pruning techniques for eliminating the redundant branches in Section 2.2.

2.1 Deep CNNs with Depthwise Separable Convolutions

The principle of depthwise separable convolutions is derived from Flattened Networks (Jin et al. (2015)) and Factorized Networks (Wang et al. (2017)). Jin et al. (2015) separate the standard 3D convolutional filter into a set of 1D convolutional filters over channel, along vertical and horizontal directions, named as flattened convolutions. The flattened convolutions can reduce numerous parameters of a CNN and enhance its accuracy slightly. Wang et al. (2017) treat the standard 3D convolutional filter as a combination of 2D spatial convolutional filters inside each channel (intra-channel convolution) and a linear projection (1×1 standard convolutional layer) which is applied to all the channels. This combination with residual connection is named as single intra-channel convolutional layer. It also can compress the CNN models and enhance the accuracy of CNN slightly.

Inspired by the above researches, Howard et al. (2017) separate the standard convolutional layer into a combination of one channel-wise group convolutional layer (depthwise convolution) and one standard convolution layer (pointwise convolution) with the kernel size of 1×1. ReLU (Nair & Hinton (2010)) and batch-normalization (Ioffe & Szegedy (2015)) are applied after each convolution. To make CNN model able to be deployed on mobile devices, they construct a novel structure of lightweight network with 13 depthwise separable convolutions, which is named as MobileNetV1 (Howard et al. (2017)). It is able to save 85% of model size with only 1.1% of accuracy drops. To further improve the information flow of MobileNetV1, Sandler et al. (2018) expand the feature map channels by inserting an extra pointwise convolution before depthwsie separable convolution. Combining with shortcut connections, it forms a novel building block called bottleneck depth-separable convolution with residuals which can better avoid information loss than depthwise separable convolution. By stacking 17 of such blocks, Sandler et al. (2018) propose MobieNetV2 that achieves accuracy improvement as well as model size reduction. Besides, MobileNetV2 is also applied as backbones of SSD (Liu et al. (2016)) and DeepLabV3 (Chen et al. (2017)) to improve inference speed for object detection and semantic segmentation tasks.

2.2 Neural Network Pruning

In the field of deep model compression, filter pruning becomes one the most popular directions because the pruned models are able to gain acceleration directly using the existing deep learning framworks, such as Caffe (Jia et al. (2014)) and PyTorch (Paszke et al. (2017)). In filter pruning, the main challenge is to recognize unimportant filters so that removing them will not cause unrecoverable catastrophic performance drops. Li et al. (2017) recognize unimportant filters by computing the $l1$ norm of filters in the pretrained large model given. He et al. (2017) utilize LASSO regression based channel selection to remove the less useful channels. Li et al. (2017) determine the importance of filters using the scale factors in batch normalization layers. After pruning the unimportant filters, these methods require retraining to compensate the losses of performance. On the other hand, Ding et al. (2019) follow another path to recognize correlated filters by K-means clustering and propose

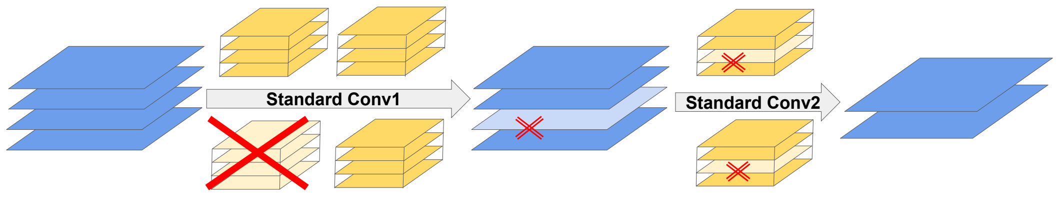

Figure 1: Illustration of the filter pruning of a standard convolution. Suppose that there are 4 filters in Conv1, pruning the third filter will remove the third channel of output feature maps. After that, the third channel in all filters in the next layer (Conv2) should also be removed.

C-SGD to force filters in the same cluster to converge to the same values. Therefore, the redundant filters can be removed without the need for retraining.

Apart from filter pruning, several work also consider pruning finer structure like intra-kernel pruning that prunes spatial locations within convolution filters. Anwar et al. (2017) use evolutionary algorithms to prune intra-kernel structures that have the least accuracy drops. Yang et al. (2018)) generate a series of intra-kernel patterns and prune them with minimal absolute summation. Furthermore, unstructured pruning treats each weight as a pruning candidate and thus becomes the finest pruning strategy. Under this category, Zhu & Gupta (2018) iteratively prune a handful of weights and retrains to recover the performance immediately. Although they can maintain the accuracy while increasing the sparsity, special libraries or hardware are required to gain acceleration, which imposes the development overhead in real applications.

3 METHODOLOGY

Standard convolution of CNN is formulated as follows. Denote $\mathbf{F}^{(i)} \in \mathbb{R}^{c_{i+1} \times c_i \times k \times k}$ as the convolution kernel of the i-th layer with k as the spatial size of the kernel. This layer takes an input feature map $\mathbf{X}^{(i)} \in \mathbb{R}^{c_i \times h_i \times w_i}$ and computes the output as

$$\mathbf{Y}^{(i)}_{c,:,:} = \sum_{j=1}^{c_i} \mathbf{X}^{(i)}_{j,:,:} * \mathbf{F}^{(i)}_{c,j,:,:}, \qquad c \in \{1 \cdots c_{i+1}\}, \tag{1}$$

where "$*$" denotes the common 2D convolution of size $k \times k$. Pruning the standard convolution (e.g., removing one kernel) only affects the input depth of the next layer. As illustrated in Fig. 1, when a kernel (filter) is deleted, the output depth (which is equivalently to the input depth of the next layer) is also decreased, and thus the kernel depth in the next layer is shortened.

The convolution can be divided into groups in a layer, resulting in the operation known as group convolution (Ciresan et al. (2011), Ioannou et al. (2017)) that is originally used in AlexNet (Krizhevsky et al. (2012)). Pruning group convolutions is a structural pruning problem where the pruned parameters should follow patterns of their positions on the input and output channels. Therefore, more structural constraints have to be considered when pruning group convolutions.

3.1 DEPTHWISE SEPARABLE CONVOLUTION AND MOBILENET ARCHITECTURE

Assume that the input depth (number of channels) is c_i in the i-th layer. Depthwise convolution divides the input as c_i groups with each group containing a single channel. The output tensor $\mathbf{M}^{(i)} = DW_{conv}(\mathbf{X}^{(i)}, \mathbf{D}^{(i)})$ is given by

$$\mathbf{M}^{(i)}_{c,:,:} = \mathbf{X}^{(i)}_{c,:,:} * \mathbf{D}^{(i)}_{c,1,:,:}, \qquad c \in \{1 \cdots c_i\}. \tag{2}$$

In practical frameworks like Pytorch, this layer is implemented using grouped convolution by setting the number of groups equaling to the number of input channels.

MobileNetV1 (Howard et al. (2017)) is proposed to utilze the depthwise convolution followed by the standard convolutions of spatial size 1×1 (referred to as pointwise convolutions). Let $Std_{conv}(\mathbf{X}, \mathbf{F})$

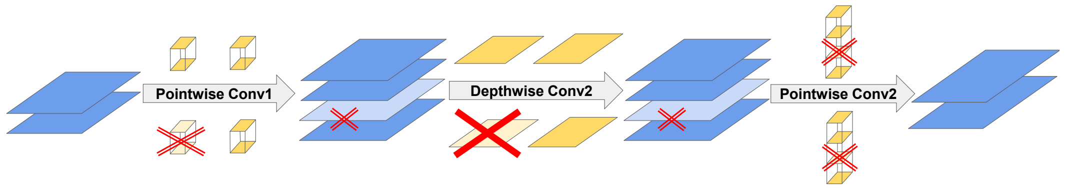

Figure 2: Pruning a depthwise separable convolution. Suppose that there are 4 filters in Depthwise Conv2, pruning the third one, as depicted in red cross, will remove the third channel of both input and output feature maps due to the parallelism in depthwise convolution. Thus, the corresponding kernels in Pointwise Conv1 and Depthwise Conv2 are also removed, as shown in double crosses.

denote the standard convolution of Eq. 1. Supposed that $\mathbf{P}^{(i)} \in \mathbb{R}^{c_{i+1} \times c_i \times 1 \times 1}$ are c_{i+1} pointwise kernels applied, MobileNetV1 then evaluates the output of the layer as follows

$$\mathbf{Y}^{(i)} = Std_{conv}(\mathbf{M}^{(i)}, \mathbf{P}^{(i)}). \tag{3}$$

The parameters in the layer are thus $(\mathbf{D}^{(i)}, \mathbf{P}^{(i)})$, where $\mathbf{D}^{(i)} \in \mathbb{R}^{c_i \times 1 \times k \times k}$ and $\mathbf{P}^{(i)} \in \mathbb{R}^{c_{i+1} \times c_i \times 1 \times 1}$ are the kernels of depthwise and pointwise convolution layers, respectively.

To further improve MobileNetV1, Sandler et al. (2018) propose to use a novel building block called bottleneck depth-separable convolution with residuals. This building block uses a shortcut connection and inserts an extra pointwise convolution that expands the input depth by a factor t before the depthwise convolution. Therefore, it contains three convolutions and computes the output as

$$\mathbf{Y}^{(i)} = Std_{conv}(DW_{conv}(Std_{conv}(\mathbf{X}^{(i)}, \mathbf{E}^{(i)}), \mathbf{D}^{(i)}), \mathbf{P}^{(i)}) + \mathbf{X}^{(i)}, \tag{4}$$

where $\mathbf{E}^{(i)} \in \mathbb{R}^{tc_i \times c_i \times 1 \times 1}$, $\mathbf{D}^{(i)} \in \mathbb{R}^{tc_i \times 1 \times k \times k}$ and $\mathbf{P}^{(i)} \in \mathbb{R}^{c_{i+1} \times tc_i \times 1 \times 1}$ are the three respective kernels, and thus its parameters are $(\mathbf{E}^{(i)}, \mathbf{D}^{(i)}, \mathbf{P}^{(i)})$.

3.2 Constrained Filter Pruning

Filter pruning is popular for structural reduction because it directly produces a thinner network that can easily fit into existing deep-learning frameworks for acceleration without needs of self-defined operations. In the following, we introduce the approach that maintains the structural consistency when pruning depthwise separable convolutions with shortcut connections.

3.2.1 Depthwise Separable Convolutions

Unlike the case of standard convolutions, pruning depthwise convolutions influences both input and output of a layer. For the i-th depthwise separable convolution with parameters $(\mathbf{D}^{(i)}, \mathbf{P}^{(i)})$, we denote the index sets of the depthwise and pointwise convolutions as $\mathbb{D}^{(i)} \subset \{0, 1, ..., c_i - 1\}$ and $\mathbb{P}^{(i)} \subset \{0, 1, ..., c_{i+1} - 1\}$. According to Eq. 2, each filter $\mathbf{D}^{(i)}_{c,1,:,:}$ operates on the input feature map $\mathbf{X}^{(i)}_{c,:,:}$. Besides, $\mathbf{X}^{(i)}_{c,:,:}$ is generated by the pointwise convolution of the $(i-1)$-th layer with the filter $\mathbf{P}^{(i-1)}_{c,:,:,:}$. Hence, we can conclude that the filter pruning pattern of the i-th depthwise convolution should be the same as that of the $(i-1)$-th pointwise convolution (i.e., $\mathbb{P}^{(i-1)} = \mathbb{D}^{(i)}$), as illustrated in Fig. 2. After pruning the filters in $\mathbf{D}^{(i)}$ and $\mathbf{P}^{(i)}$, similar to the pruning of standard convolutions, the corresponding kernels in the next layer $(i+1)$ have to be removed as well. [1]

The structural constraints are extended to the depth-separable convolution blocks in MobileNetV2 (Sandler et al. (2018)) with the parameters $(\mathbf{E}^{(i)}, \mathbf{D}^{(i)}, \mathbf{P}^{(i)})$ as defined in Eq. 4. Similarly, we denote the index sets to be $\mathbb{E}^{(i)}$, $\mathbb{D}^{(i)}$ and $\mathbb{P}^{(i)}$. Because the input feature maps of the depthwise convolution are from the extra 1×1 convolution layer, their filter pruning indices should be the same. In other words, we have $\mathbb{E}^{(i)} = \mathbb{D}^{(i)}$ for all such blocks in MobileNetV2 (Sandler et al. (2018)).

[1]Particularly, in MobileNetV1 (Howard et al. (2017)), RGB images are converted into 32-channel feature maps via standard convolutions in the first layer. Thus we have $\mathbb{F}^{(1)} = \mathbb{D}^{(2)}$ as the starting point for pruning.

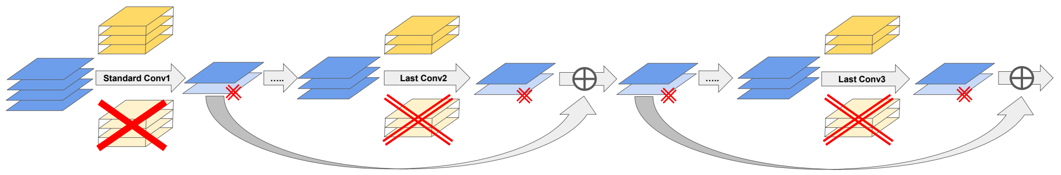

Figure 3: Illustration of pruning residual blocks following the strategy in (Ding et al. (2019)). Supposed that there are 2 filters in Standard Conv1 which is followed by 2 residual blocks. Let Last Conv2 and Last Conv3 be the last layers of the two residual blocks, respectively. Pruning the second filter in Conv1 will force the second filters in Last Conv2 and Last Conv3 to be removed.

3.2.2 Shortcut Connections

A shortcut connection that sums the learned residuals and the stem features introduces dependencies between the two layers that are linked. We call the layer producing the stem features the *pacesetter*, and the last layer of the residual block the *follower*. To maintain the validation of the information flow after filter pruning, the pacesetter and the follower must have the same pruning patterns. Li et al. (2017) skips the pruning of pacesetters and followers and only prune other internal layers in the residual blocks. Accordingly, the input and output feature maps remain to have the same number of channels as the original model, and this limits the amount of prunable model size. Liu et al. (2017) and He et al. (2017) insert a sampler layer before the first internal layer in the residual block to reduce the channels of the input feature maps. Though they can remove the corresponding kernels in the first internal layer, the filters of pacesetters and followers are un-pruned, which still limits the possibility of further efficiency boosting.

In our approach, to fully boost the efficiency, we follow the strategy in (Ding et al. (2019)) that allows to prune the whole network without sidestepping these troublesome layers. Starting from the beginning of MobileNetV2 (Sandler et al. (2018)), let m be the layer index of the non-residual block just before a sequence of n bottleneck depth-separable convolution residual blocks, as shown in Fig. 3. Without loss of generality, we assume that the m-th layer is a standard convolution layer, which is usually the case in MobileNetV2 (Sandler et al. (2018)). Each of the following n residual blocks has a pointwise convolution as its last layer within the block. The m-th layer produces the stem feature maps, and thus is called the pacesetter; the following blocks' pointwise convolutions are then the followers. Therefore, for all such structures starting with a standard convolution layer and followed by residual blocks, we can derive the constraints as shown below:

$$\mathbb{F}^{(m)} = \mathbb{P}^{(m+1)} = \mathbb{P}^{(m+2)} = ... = \mathbb{P}^{(m+n)}. \tag{5}$$

3.3 Gradual Pruning with Multiple Stages

In the above, we have derived the filter pruning constraints implied by the network structures. In this section, we describe our pruning approach for MobileNets. Gradual pruning (Zhu & Gupta (2018)) is a simple but effective technique that iteratively prunes a small amount of weights with low absolute values untill the target sparsity is reached. The original approach of (Zhu & Gupta (2018)) is used for unstructured pruning that removes weights without following a regular network structure. Therefore, no direct speedup can be attained when implementing the pruned model to existing deep learning frameworks. In this work, we apply the gradual pruning principle to filter removing following the derived constraints. Besides, we extend the principle to multi-stage gradual pruning. The original gradual pruning approach is a special case of our approach of a single stage, and our multi-stage gradual pruning method provides a smoother track of pruning for handling the depthwise separable structure.

Given a pretrained model, we first investigate all the filter pruning constraints and cluster the kernels having to share the same pruning patterns. Supposed that there are G clusters of kernels and denote $\mathbb{W}^g$ as the set containing all the kernels in the g-th cluster. Let $\mathbb{I}^g$ be its index set of pruned filters, where $g \in \{1, 2, ..., G\}$. To prune the given model from sparsity 0.0 to a final sparsity s_f, we split this process into S stages and apply gradual pruning in each stags. Gradual pruning aims to prune a little amount of weights every Δt iterations as the model is trained to maintain the performance while increasing the sparsity. The amount of weights being removed in each pruning iteration is computed by a cubic function to interpolate the beginning sparsity and end sparsity in each stage. Supposed the starting iteration for a stage is t_0, the sparsity after every Δt iterations is as follows:

$$s_t = s_b + (s_e - s_b)(1 - \frac{t}{n\Delta t}), \text{ for } t \in \{0, \Delta t, ..., n\Delta t\}, \tag{6}$$

where s_b and s_e are the starting and end sparsity for each stage, and n is the number of pruning iterations. When a pruning iteration proceeds, we compute the importance score for all candidates by summing the absolute values of filters and remove the ones with smaller scores. Details can be found in Algorithm 1.

Algorithm 1: Multistage Gradual Pruning

```
Input: Kernel cluster sets $\{\mathbb{W}^1, \mathbb{W}^2, ..., \mathbb{W}^G\}$, final sparsity $s_f$, number of stages $S$, pruning epoch
       $E_p$ and finetune epoch $E_f$
Output: Index sets $\{\mathbb{I}^1, \mathbb{I}^2, ..., \mathbb{I}^G\}$ of the pruned filters
Data: Training set $\mathcal{D}$
1  for $s$ in $0, 1, ..., S-1$ do
2      Compute the beginning sparsity $s_b = s\frac{s_f}{S}$ and end sparsity $s_e = (s+1)\frac{s_f}{S}$
3      Determine the number of iterations in a pruning epoch $T = E_p \times \#iters\_per\_epoch$
4      for $t$ in $0, 1, ..., T-1$ do
5          if $t$ is multiple of $\Delta t$ then
               // Prune a small amount of the model
6              Determine current sparsity $s_t$ by Eq. 6
7              for $g$ in 1, 2, ..., $G$ do
                   // AbaSum() sums all tensor elements' absolute values
8                  For all filter $c$, compute the score $\sum_{\mathbf{W} \in \mathbb{W}^g} \text{AbsSum}(\mathbf{W}_{c,:,:,:})$
9                  Add the indices of filters with the smallest score into $\mathbb{I}^g$ untill reaching sparsity $s_t$
10                 Let $\mathbb{T} = \{\mathbf{W}_{c,:,:,:} | \forall c \in \mathbb{I}^g, \mathbf{W} \in \mathbb{W}^g\}$ be the set of all pruned filters
11                 Set all filters in $\mathbb{T}$ to 0.0
12         Train the model using a batch of data in $\mathcal{D}$ without updating filters in $\mathbb{T}$
       // Finetune the model to regain performance
13     Train the model for $E_f$ epochs on $\mathcal{D}$ without updating filters in $\mathbb{T}$
```

4 Experiments and Results

In this section, we show the experimental studies to verify the effectiveness of our approach.

4.1 Experimental Settings

To evaluate the performance on pruning lightweight models, we conduct experiments by applying our proposed method on MobileNetV1 and MobileNetV2 using the following two benchmarks:
CIFAR10 (Krizhevsky et al. (2009)) is a widely used benchmark for image classification. It contains sixty thousands 32×32 color images of objects seen in daily lives, such as animals and vehicles. Each object has 6,000 images. To verify the model, they are split into 50,000 images as the training set and 10,000 images as the test set.
SVHN (Netzer et al. (2011)) is a popular benchmark for recognizing digits in natural scene images, like MNIST database (LeCun et al. (1998)). It consists of 99,289 digits images with the resolution equal to 32×32, which are split into 73,257 images for training set and 26,032 for testing.

We first train baseline models on CIFAR10 and SVHN for 300 epochs using SGD optimizers with learning rates 0.1 and 0.01, respectively; for both datasets, the momentum is 0.9 and weight-decay is 1×10^{-4}. The learning rates decay 0.1 every 80 epochs. In each stage of gradual pruning, we use 4 epochs for pruning with the frequency of 200 iterations and 26 epochs for fine-tuning. Our metrics include relative accuracy drops and execution time speedups between the baseline and pruned models. The execution time is measured by running 500 forward passes with batch size 512 on the network models to record the elapsed time. Our method is implemeneted by using PyTorch (Paszke et al. (2017)) and evaluated on a Nvidia Geforce GTX 1080Ti GPU.

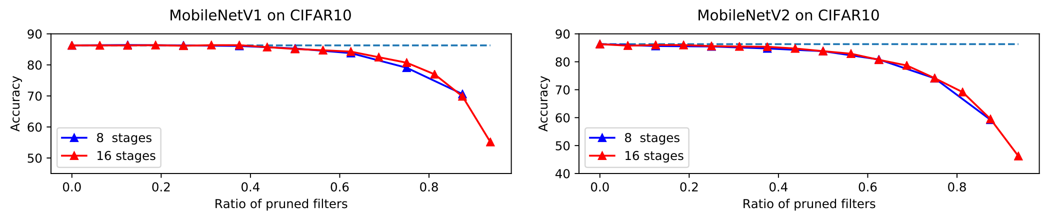

Figure 4: Accuracy of the pruned MobileNets on CIFAR10 with 8 and 16 stages of gradual pruning.

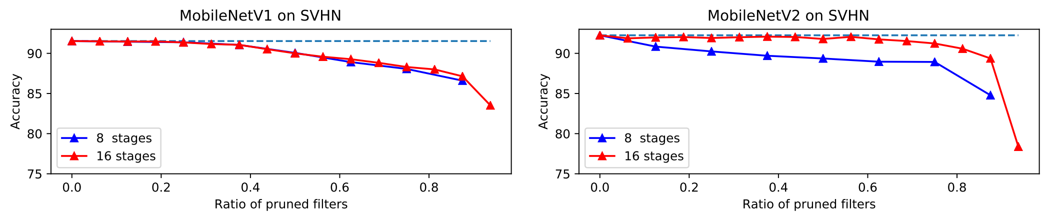

Figure 5: Accuracy of the pruned MobileNets on SVHN with 8 and 16 stages of gradual pruning.

4.2 Ablation Study

Because our method further extends the gradual pruning into multistage gradual pruning, we conduct ablation study on how the number of stages affects the performance. When the number of stages increases, the pruning ratio in each stage decreases, which results in a finer filter pruning procedure. We compare multistage gradual pruning with 8 and 16 stages, which prunes $1/8$ and $1/16$ of filters in one stage, respectively. In Fig. 4, the performance of 8 and 16 stages on the CIFAR10 dataset are similar, and note that 8-stage pruning already performs well on maintaining the accuracy when the ratio of pruned filters increases. However, 16 stages pruning can still maintain accuracy better on around 0.8 of filters pruned than the 8-stage setting. In Fig. 5 (on the SVHN dataset), we find that the 8-stage pruning suffers from accuracy drops when the sparsity is increased, but the 16-stage pruning manages to maintain the performance significantly better. We owe this to the reason that it is easier to recover performance by fine-tuning when performing a finer pruning procedure.

To further investigate the influence of fine- and coarse-grain pruning, we take the model (on the SVHN dataset) with $1/8$ filters pruned in two stages from the 16-stage pruning. Then, we further prune $1/8$ filters of this model in only one stage. After that, the pruning ratio becomes $2/8$ with an accuracy of 91.70%. Compared with the 8-stage pruning's model with the same pruning ratio that has the accuracy of 90.24%, we conjecture that finer pruning in the beginning stage helps to find a better local minimum, and after that coarse-grain pruning can be applied to speed up the pruning process. In another setting, we take the model with $6/8$ filters pruned from the 8-stage pruning and further prune $1/8$ filters in two stages with the 16-stage pruning. The resulted model has $7/8$ filters pruned with the accuracy of 88.07%, which is better than the model with the same pruning ratio in 8-stage pruning having the accuracy of 84.78%. Hence, fine grain pruning process is also capable of maintaining accuracy even when the model is already stuck in a worse local optimum. With these observations, our multistage gradual pruning could potentially recover the performance by switching to finer pruning at any stage when the current pruning stage suffers from severe accuracy drops.

4.3 Efficiency Improvement

We compress the trained MobileNets on CIFAR10 and SVHN using 16 stages, i.e. pruning $1/16$ of filters in each stage, and report the performance on the models with $1/4$ filters pruned, as shown in Table 1 and Table 2. On both CIFAR10 and SVHN datasets, our method can maintain the accuracy with less than 1% relative accuracy drops. The results indicate that even in lightweight models, there still exist redundant weights that can be removed with neglectable drops of accuracy. After pruning $1/4$ filters of the model, the FLOPs can be reduced to around $1 - (3/4)^2 = 43.75\%$. However,

Table 1: Accuracy of MobileNetV1 with $1/4$ of filters pruned on CIFAR10 and SVHN.

MobileNetV1	**Base Top1 Acc**	**Pruned Top1 Acc**	**Rel. ↓ %**	**Speedup ↑ %**
CIFAR10	86.28	86.15	0.15	40.66
SVHN	91.53	91.36	0.19	

Table 2: Accuracy of MobileNetV2 with $1/4$ of filters pruned on CIFAR10 and SVHN.

MobileNetV2	**Base Top1 Acc**	**Pruned Top1 Acc**	**Rel. ↓ %**	**Speedup ↑ %**
CIFAR10	86.31	85.61	0.81	67.74
SVHN	92.25	91.91	0.37	

considering additional memory access overheard on real machines, our approach still achieves 40% and 67% of speedups for MobileNetV1 and V2, respectively.

5 Conclusions and Future Work

In this paper, a multi-stage gradual pruning approach for depthwise separable convolution networks is proposed. Redundancy of the lightweight model, MobileNet, can be further exploited in a smoother multi-stage manner of pruning. In our methods, we prune filters considering structural constraints so that we are able to fully boost the efficiency. Our experiments show that we can maintain the accuracy better when we use finer pruning ratios and more stages. We also show that extra efficiency can indeed be acquired by pruning lightweight models. In the future, we plan to apply our approach to the depthwise separable convolution structures found by NAS for further improving the inference efficiency.

References

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HylVB3AqYm`.

Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. URL `https://docs.nvidia.com/deeplearning/sdk/cudnn-developer-guide/index.html`.

Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4943–4953, 2019.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. URL `http://arxiv.org/abs/1503.02531`.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Yani Ioannou, Duncan Robertson, Roberto Cipolla, and Antonio Criminisi. Deep roots: Improving cnn efficiency with hierarchical filter groups. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1231–1240, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. Flattened convolutional neural networks for feedforward acceleration. In *International Conference on Learning Representations Workshops*, 2015. URL `https://arxiv.org/abs/1412.5474`.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=rJqFGTslg`.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 19–34, 2018.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer, 2016.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2736–2744, 2017.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

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

Min Wang, Baoyuan Liu, and Hassan Foroosh. Factorized convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 545–553, 2017.

Maurice Yang, Mahmoud Faraj, Assem Hussein, and Vincent Gaudet. Efficient hardware realization of convolutional neural networks using intra-kernel regular pruning. In *2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL)*, pp. 180–185. IEEE, 2018.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. In *International Conference on Learning Representations Workshops*, 2018. URL `https://openreview.net/pdf?id=Sy1iIDkPM`.