ThiNet: Pruning CNN Filters for a Thinner Net

Jian-Hao Luo, Hao Zhang, Hong-Yu Zhou, Chen-Wei Xie,
Jianxin Wu, *Member, IEEE*, and Weiyao Lin, *Senior Member, IEEE*

Abstract—This paper aims at accelerating and compressing deep neural networks to deploy CNN models into small devices like mobile phones or embedded gadgets. We focus on filter level pruning, i.e., the whole filter will be discarded if it is less important. An effective and unified framework, ThiNet (stands for "Thin Net"), is proposed in this paper. We formally establish filter pruning as an optimization problem, and reveal that we need to prune filters based on statistics computed from its next layer, not the current layer, which differentiates ThiNet from existing methods. We also propose "gcos" (Group COnvolution with Shuffling), a more accurate group convolution scheme, to further reduce the pruned model size. Experimental results demonstrate the effectiveness of our method, which has advanced the state-of-the-art. Moreover, we show that the original VGG-16 model can be compressed into a very small model (ThiNet-Tiny) with only 2.66 MB model size, but still preserve AlexNet level accuracy. This small model is evaluated on several benchmarks with different vision tasks (e.g., classification, detection, segmentation), and shows excellent generalization ability.

Index Terms—Convolutional neural networks, filter pruning, deep learning, model compression

1 Introduction

In the past few years, we have witnessed a rapid development of deep neural networks in the field of computer vision, from basic image classification tasks such as the ImageNet recognition challenge [1], [2], [3], to some more advanced applications, e.g., object detection [4], semantic segmentation [5], style transfer [6] and many others. Deep neural networks have achieved state-of-the-art performance in these fields compared with traditional methods based on manually designed visual features.

In spite of its great success, a typical deep model is hard to be deployed on resource constrained devices, e.g., mobile phones or embedded gadgets. A resource constrained scenario means a computing task must be accomplished with limited resource supply, such as computing time, storage space, battery power, etc. One of the main issues of deep neural networks is its huge computational cost and storage overhead, which constitutes a serious challenge for a mobile device. For instance, the VGG-16 model [2] has 138.34 million parameters, taking up more than 500 MB storage space, and needs 30.94 billion float point operations (FLOPs) to classify a single image. Such a cumbersome model can easily exceed the computing limit of small devices. Thus, network compression has drawn a significant amount of interest from both academia and industry.

- *J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, and J. Wu are with the National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: {luojh, zhangh, zhouhy, xiecw, wujx}@lamda.nju.edu.cn.*
- *W. Lin is with the Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai 200000, China. E-mail: wylin@sjtu.edu.cn.*

Manuscript received 4 Dec. 2017; revised 5 June 2018; accepted 18 July 2018. Date of publication 19 July 2018; date of current version 12 Sept. 2019. (Corresponding author: Jianxin Wu.)
Recommended for acceptance by N. Vasconcelos.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPAMI.2018.2858232

Pruning is one of the most popular methods to reduce network complexity, which has been widely studied in the model compression community. In the 1990s, LeCun et al. [7] had observed that several unimportant weights can be removed from a trained network with negligible loss in accuracy. A similar strategy was also explored in [8]. This process resembles the biological phenomena in mammalian brain, where the number of neuron synapses has reached the peak in early childhood, followed by gradual pruning during its development. However, these methods are mainly based on the second derivative, thus are not applicable for today's deep model due to expensive computation costs.

Recently, Han et al. [9] introduced a simple pruning strategy: all connections with weights below a threshold are removed, followed by fine-tuning to recover its accuracy. This iterative procedure is performed several times, generating a very sparse model. However, such a non-structured sparse model cannot be supported by off-the-shelf libraries, thus specialized hardwares and softwares are needed for efficient inference, which is difficult and expensive in real-world applications. On the other hand, the non-structured random connectivity ignores cache and memory access issues. As indicated in [10], due to the poor cache locality and jumping memory access caused by random connectivity, the *practical acceleration* is very limited (sometimes even slows down), even though the actual sparsity is high.

To avoid the limitations of non-structured pruning mentioned above, this paper argue that *filter level* pruning will be a better choice. The benefits of removing the whole unimportant filter have a great deal: 1) The pruned model has no difference in network structure, thus it can be perfectly supported by any off-the-shelf deep learning library. 2) Memory footprint will be reduced dramatically. Such memory reduction comes not only from the model itself, but also from the intermediate activation, which is rarely considered in previous studies. 3) Since the pruned network structure has not been damaged, it can be further compressed

and accelerated by other compression methods, e.g., the parameter quantization approach [11]. 4) More vision tasks, such as object detection or semantic segmentation, can be accelerated greatly using the pruned model.

We propose a unified framework, ThiNet (which stands for *"Thin Net"*), to prune the unimportant filters to simultaneously accelerate and compress CNN models in both training and test stages with minor performance degradation. With our pruned network, some important transfer learning tasks such as object detection or fine-grained recognition can run much faster (in both training and inference), especially in small devices. Our main insight is that we establish *a well-defined optimization problem*, which shows that *whether a filter can be pruned depends on the outputs of its* next *layer, not its own layer*. This novel finding differentiates ThiNet from existing methods which prune filters using statistics calculated from their own layer.

One of the most significant advantages of filter level pruning is its compatibility: the model structure has not been destroyed, hence the pruned network can be further compressed with other methods. We proposed a more accurate group convolution scheme, gcos (Group COnvolution with Shuffling), to further reduce model size. Traditional group convolution will face a severe "information blocking" problem: *different groups cannot communicate with each other*. In gcos, we use 1×1 convolution to shuffle the information of different groups, hence the accuracy will be preserved.

We then compare the proposed method with other state-of-the-art criteria. Experimental results show that our approach is significantly better than existing methods, especially when the compression rate is relatively high. We evaluate ThiNet on the large-scale ImageNet classification task. ThiNet achieves $3.23\times$ FLOPs reduction on the VGG-16 model [2], with 1.76 percent top-1 and 0.6 percent top-5 accuracy drop. The ResNet-50 model [3] has less redundancy compared with classic CNN models. ThiNet can still reduce $2.26\times$ FLOPs and $2.06\times$ parameters with 3.27 percent top-1 and 1.21 percent top-5 accuracy drop.

To explore the limits of ThiNet, we prune VGG-16 into two small but accurate models with the help of gcos, ThiNet-Tiny and ThiNet-Small. ThiNet-Tiny only takes up 2.66 MB space, but still preserves AlexNet level accuracy. ThiNet-Small is much more accurate, whose top-1 accuracy on ImageNet is 62.97 percent while its size is only 4.67 MB. Further experiments on several vision tasks (e.g., classification, detection, segmentation) demonstrate the excellent generalization ability of ThiNet models, which achieve the best trade-off between model size and accuracy.

The rest of this paper is organized as follows. Related work is discussed in Section 2. We will introduce the details of ThiNet in Section 3. Section 4 presents the experiments to compare ThiNet with others. Finally, we conclude this paper in Section 5.

2 Related Work

Many researchers have found that deep models suffer from heavy over-parameterization. For example, Denil et al. [12] demonstrated that a network can be efficiently reconstructed with only a small subset of its original parameters. However, this redundancy seems *necessary* during model training, since the highly non-convex optimization is hard to be solved with current techniques [13], [14]. Hence, there is a great need to reduce model size *after* its training.

Some methods have been proposed to pursue a balance between model size and accuracy. Han et al. [9] proposed an iterative pruning method to remove the redundancy in deep models. Their main insight is that small-weight connectivity below a threshold should be discarded. In practice, this can be aided by applying ℓ_1 or ℓ_2 regularization to push connectivity values to become smaller. The major weakness of this strategy is the loss of universality and flexibility, thus seems to be less practical in real applications.

In order to avoid these weaknesses, some attention has been focused on the group-wise sparsity. Lebedev and Lempitsky [15] explored group-sparse convolution by introducing the group-sparsity regularization to the loss function, then some entire groups of weights will shrink to zeros and can be removed. Mao et al. [16] explored different granularity of sparsity from irregular connection pruning to regular filter pruning. Wen et al. [10] proposed the Structured Sparsity Learning (SSL) method to regularize filter, channel, filter shape and depth structures. In spite of their success, the original network structure has been destroyed. As a result, some dedicated libraries are needed to obtain inference speed-up in real applications.

In line with our work, some filter level pruning strategies have been explored, too. The core is to evaluate neuron importance, which has been widely studied in the community [17], [18], [19], [20], [21]. A simple method is based on the magnitude of weights. Li et al. [19] measured the importance of each filter by calculating its absolute weight sum. Another practical criterion is to measure the sparsity of activations after the ReLU function [22]. Hu et al. [20] believed that if most outputs of some neurons are zero, these activations should be expected to be redundant. They compute the Average Percentage of Zeros (APoZ) of each filter as its importance score. These two criteria are simple and straightforward, but not directly related to the final loss. Inspired by this observation, Molchanov et al. [21] adopted Taylor expansion to approximate the influence to loss function induced by removing each filter.

Beyond pruning, there are also other strategies to obtain small CNN models. One popular approach is parameter quantization [11], [23], [24], [25]. Low-rank approximation is also widely studied [13], [26]. Note that these methods are complementary to filter pruning, which can be combined with ThiNet for further improvement.

3 ThiNet

We exploit a filter level pruning method to reduce model complexity with the goal of minimizing activation reconstruction error. In this section, we present the overall ThiNet framework first. Our method consists of two major parts: pruning and post-processing. Preliminary version of portions of this work have been published in [27], which is briefly introduced in Section 3.2.3. Then, we introduce the improved version in Section 3.2.4.

3.1 The ThiNet Framework

Fig. 1 shows the overall pipeline of ThiNet. Given a pre-trained CNN model, we first use the proposed novel method

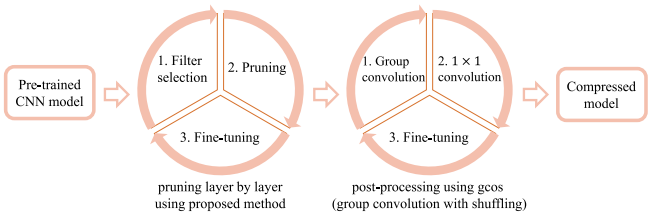


Fig. 1. The overall ThiNet pipeline, which is divided into two major parts. Given a pre-trained CNN model, it is pruned layer by layer using our proposed filter selection method, followed by post-processing to further reduce model parameters.

to prune several unimportant filters. Since we focus on filter level pruning, our pruned model can be further compressed by other methods (the post-processing part). Here, we use group convolution to further reduce model parameters. However, if we simply adopt this strategy in the pruned model, the accuracy will be damaged greatly. Hence, we use 1×1 convolution (also known as pointwise convolution) to tackle the information blocking problem of traditional group convolution. And we call this method "gcos": *Group COnvolution with Shuffling*.

3.2 Part 1: Pruning

We first focus on the pruning part, which can be summarized in one sentence: evaluate the importance of each neuron, remove those unimportant ones, and fine-tune the whole network.

3.2.1 Overview of ThiNet Pruning

The pruning processing is illustrated in Fig. 2. Starting from a pre-trained model, we prune it layer by layer with a *predefined compression rate*. We summarize our pruning framework as follows:

1) *Filter selection.* Unlike existing methods that use layer i's statistics to guide the pruning of layer i's filters, we use layer $i+1$ to guide the pruning in layer i. The key idea is: *if we can use a subset of channels in layer* $(i+1)$*'s input to approximate the output in layer* $i+1$*, the other channels can be safely removed* from the input of layer $i+1$. Note that one channel in layer $(i+1)$'s input is produced by one filter in layer i, hence we can safely prune the corresponding filter in layer i.
2) *Pruning.* Weak channels in layer $(i+1)$'s input and their corresponding filters in layer i are pruned away, leading to a much smaller model. Note that the pruned network has exactly the same structure as the original network, but with fewer filters and channels. In other words, the original wide network is becoming much thinner. That is why we call our method "ThiNet".
3) *Fine-tuning.* Fine-tuning is a necessary step to recover model accuracy damaged by filter pruning. But it will take very long time for large datasets and complex models. To reduce training time, we fine-tune one or two epochs after the pruning of one layer. In order to get an accurate model, additional epochs are carried out after all layers have been pruned.
4) *Iterate to step 1 to prune the next layer.*

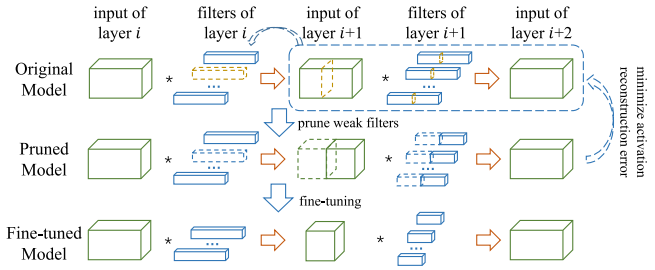


Fig. 2. Illustration of ThiNet's *pruning* part. We first focus on the dotted box to determine several weak channels and their corresponding filters (highlighted in yellow in the first row). These channels (and their associated filters) have little contribution to the overall performance, thus can be discarded, leading to a pruned model. Finally, the network is fine-tuned to recover its accuracy. (This figure is best viewed in color.)

Now we will focus on the dotted box in Fig. 2 to introduce our data-driven channel selection method, which determines the channels (and their associated filters) that are to be pruned away. We use a triplet $\langle \mathcal{I}_i, \mathcal{W}_i, * \rangle$ to denote the convolution process in layer i, where $\mathcal{I}_i \in \mathbb{R}^{C \times H \times W}$ is the input tensor, which has C channels, H rows and W columns. And $\mathcal{W}_i \in \mathbb{R}^{D \times C \times K \times K}$ is a set of filters with $K \times K$ kernel size, which outputs a new tensor with D channels.

Our goal is to remove some unimportant filters in $\mathcal{W}_i$. Note that, if a filter in $\mathcal{W}_i$ is removed, its corresponding channel in $\mathcal{I}_{i+1}$ and $\mathcal{W}_{i+1}$ are also discarded. However, since the number of filters in layer $i+1$ is not changed, the size of its output tensor, i.e., $\mathcal{I}_{i+2}$, will remain exactly the same. Inspired by this observation, we believe that if we remove filters that have little influence on $\mathcal{I}_{i+2}$ (which is also the output of layer $i+1$), it will have little influence on the overall performance, too. In other words, minimizing the reconstruction error of $\mathcal{I}_{i+2}$ is closely related to the network's classification performance.

3.2.2 Collecting Training Examples

In order to determine which channel can be safely removed, a training set used for neuron importance evaluation must be collected. As illustrated in Fig. 3, an element, denoted by y, is randomly sampled from the tensor $\mathcal{I}_{i+2}$ (before ReLU). A corresponding filter $\widehat{\mathcal{W}} \in \mathbb{R}^{C \times K \times K}$ and sliding window $x \in \mathbb{R}^{C \times K \times K}$ (after ReLU) can also be determined according to its location. The convolution operation is computed with a corresponding bias b as follows:

$$y = \sum_{c=1}^{C} \sum_{k_1=1}^{K} \sum_{k_2=1}^{K} \widehat{\mathcal{W}}_{c,k_1,k_2} \times x_{c,k_1,k_2} + b\,. \tag{1}$$

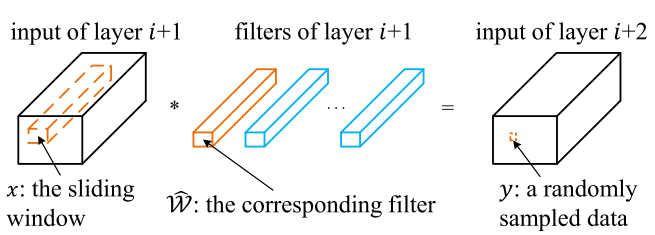


Fig. 3. Illustration of data sampling and relationship among variables. We first randomly sample an element y from the activation tensor of layer $i+1$ with random spatial location and random channel index. According to the spatial location and channel index of y, the corresponding filter $\widehat{\mathcal{W}}$ and sliding window x can also be determined.

Now, if we further define:

$$\hat{x}_c = \sum_{k_1=1}^{K}\sum_{k_2=1}^{K} \widehat{\mathcal{W}}_{c,k_1,k_2} \times x_{c,k_1,k_2}, \tag{2}$$

Eq. (1) can be simplified as:

$$\hat{y} = \sum_{c=1}^{C} \hat{x}_c, \tag{3}$$

in which $\hat{y} = y - b$. It is worthwhile to keep in mind that $\hat{x}$ and $\hat{y}$ are random variables whose instantiations require fixed spatial locations. A key observation is that channels in $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_C)$ is independent: $\hat{x}_c$ *only depends* on $x_{c,:,:}$, which has no dependency relationship with $x_{c',:,:}$ if $c' \neq c$.

In other words, if we can find a subset $S \subset \{1, 2, \ldots, C\}$ and the equality

$$\hat{y} = \sum_{c \in S} \hat{x}_c, \tag{4}$$

always holds, then we do not need any $\hat{x}_c$ if $c \notin S$ and these variables can be *safely removed without changing the CNN model's result at all*.

Of course, Eq. (4) cannot always be true for all instances of the random variables $\hat{\mathbf{x}}$ and $\hat{y}$. However, we can manually extract instances of them to find a subset S such that Eq. (4) is approximately correct.

Given an input image, we first apply the CNN model in the forward run to find the input and output of layer $i+1$. Then, for any feasible (c, k_1, k_2) triplet, we can obtain a C-dimensional vector variable $\hat{\mathbf{x}} = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_C\}$ and a scalar value $\hat{y}$ using Eqs. (1), (2), and (3). Since $\hat{\mathbf{x}}$ and $\hat{y}$ can be viewed as random variables, more instances can be sampled by choosing different input images, different channels, and different spatial locations.

Finally, we obtain a training matrix $\mathbf{X} \in \mathbb{R}^{m \times C}$ and a target vector $\mathbf{y} \in \mathbb{R}^{m \times 1}$, where m is the number of training examples (product of the number of the images and locations). Then, we use these training examples and their corresponding targets to determine which filter should be preserved or discarded.

3.2.3 Filter Selection and Weight Rescaling

Now, given a set of m training examples (matrix $\mathbf{X}$ and target vector $\mathbf{y}$), we formulate the original channel selection problem as the following optimization problem:

$$\begin{aligned} &\arg\min_{S} \sum_{i=1}^{m} \left(\mathbf{y}_i - \sum_{j \in S} \mathbf{X}_{i,j} \right)^2 \\ &\text{s.t.} \quad |S| = C \times r, \quad S \subset \{1, 2, \ldots, C\}. \end{aligned} \tag{5}$$

Here, $|S|$ is the number of elements in a subset S, and r is a predefined compression rate (i.e., how many channels are preserved). Solving Eq. (5) is equivalent to minimizing the reconstruction error. In our scenario, minimum reconstruction error means minimum information loss, hence the accuracy will not be damaged greatly. However, solving Eq. (5) is a NP hard problem. We use a greedy strategy presented in our preliminary conference paper [27] to address this problem. We add one element to S at a time, and choose the channel leading to the smallest objective value in the current iteration.

Obviously, this greedy solution is sub-optimal. But, the gap can be compensated by subsequent fine-tuning. After obtaining the subset S, we can safely remove the nth channel in each filter of layer $i+1$ if $n \notin S$. The corresponding filters in the previous layer i can be pruned, too.

We further minimize the reconstruction error (c.f. Eq. (5)) by scaling the channel weights, which is formulated as:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{m} (\mathbf{y}_i - \widehat{\mathbf{X}}_i \mathbf{w})^2, \tag{6}$$

where $\widehat{\mathbf{X}}_i = \mathbf{X}_{i,S}$ indicates the training examples after channel selection. Eq. (6) is a classic linear regression problem, which has a unique closed-form solution using the ordinary least squares approach:

$$\hat{\mathbf{w}} = (\widehat{\mathbf{X}}^{\mathrm{T}} \widehat{\mathbf{X}})^{-1} \widehat{\mathbf{X}}^{\mathrm{T}} \mathbf{y}. \tag{7}$$

Each element in $\hat{\mathbf{w}}$ can be interpreted as a scaling factor of its corresponding filter channel such that $\mathcal{W}_{:,i,:,:} = \hat{w}_i \mathcal{W}_{:,i,:,:}$. From another point of view, this scaling operation provides a better initialization for fine-tuning, hence the network is more likely to reach higher accuracy.

Algorithm 1. A Greedy Algorithm for Minimizing Eq. (8)

Input: Training set $\mathbf{X}$, $\mathbf{y}$, and compression rate r
Output: The subset of preserved channels: S and a corresponding scaling vector $\mathbf{w}$

$S \leftarrow \emptyset$; $I \leftarrow \{1, 2, \ldots, C\}$; $\hat{\mathbf{y}} \leftarrow \mathbf{y}$;
while $|S| < C \times r$ **do**
  $min\_value \leftarrow +\infty$;
  for each item $i \in I$ and $i \notin S$ **do**
    $\hat{\mathbf{w}} \leftarrow (\mathbf{X}_{:,i}^{\mathrm{T}} \mathbf{X}_{:,i})^{-1} \mathbf{X}_{:,i}^{\mathrm{T}} \hat{\mathbf{y}}$;
    $value \leftarrow \|\hat{\mathbf{y}} - \mathbf{X}_{:,i} \hat{\mathbf{w}}\|_2^2$;
    if $value < min\_value$ **then**
      $min\_value \leftarrow value$; $min\_i \leftarrow i$;
    end if
  end for
  move min_i into S;
  $\mathbf{w} \leftarrow (\mathbf{X}_{:,S}^{\mathrm{T}} \mathbf{X}_{:,S})^{-1} \mathbf{X}_{:,S}^{\mathrm{T}} \mathbf{y}$;
  $\hat{\mathbf{y}} \leftarrow \mathbf{y} - \mathbf{X}_{:,S} \mathbf{w}$;
end while

3.2.4 Improvement: Integrate Weight Rescaling into Selection

And now, here comes another question: can we combine filter selection and weight rescaling to achieve better performance? Or, can we use weight scaling to guide the pruning process? Let us revisit the optimization goal. If we can assign a scaling factor $\hat{w}_j \in \mathbb{R}$ to each filter channel, Eq. (5) is rewritten as:

$$\begin{aligned} &\arg\min_{S} \sum_{i=1}^{m} \left(\mathbf{y}_i - \sum_{j \in S} \hat{w}_j \mathbf{X}_{i,j} \right)^2 \\ &\text{s.t.} \quad |S| = C \times r, \quad S \subset \{1, 2, \ldots, C\}. \end{aligned} \tag{8}$$

In this new setup, only those channels which have minimal reconstruction loss after rescaling are preserved. And, we only need to revise our greedy algorithm with very little change. Of course, these scaling factors are not globally optimal. Hence, we need to perform another least squares after selection as we have done in Eq. (7). Algorithm 1 summarizes our final pruning strategy.

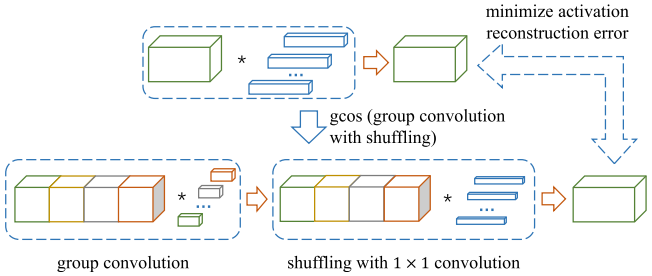


Fig. 4. Illustration of gcos: The filter weights are first divided into g groups (highlighted in different colors), followed by 1×1 convolution to solve the information blocking problem. These 1×1 filters are initialized using the design idea of "minimizing activation reconstruction error". (This figure is best viewed in color.)

In the experiment, we find that this new strategy shows better performance than the strategy in [27] with the cost of only negligible computing time increasing. With the preserved channel index set S and scaling vector $\mathbf{w}$, we can safely remove those weak filters in layer i. As for layer $i+1$, we first discard corresponding channels to reduce model size, and then rescale the filter weights using $\mathbf{w}$. After that, the pruned model can be fine-tuned.

3.3 Part 2: Post-Processing

One of the great advantages of filter level pruning is that we do not alter the network structure, hence it can be further compressed or accelerated by other compression methods. This post-processing can be finished via any current compression technique, e.g., the quantization methods [11], [23], [24], [25]. In order to match the requirements of being able to use any off-the-shelf deep learning library, we focus on group convolution.

Group convolution was first proposed in AlexNet [1], which aims at solving the limited GPU memory problem. At that time, AlexNet is too big to fit on one GPU, hence they spread it across two GPUs. In group convolution, input tensors and filter weights are divided into several groups, convolution operations are only performed within the same group, i.e., the inputs of group 1 can only do convolution calculations with filters of group 1.

However, it may lead to a severe problem: *different groups cannot communicate with each other*! In this paper, we call it the "information blocking" problem, which may greatly harm model accuracy. In AlexNet, in order to tackle the information blocking problem, GPUs can communicate at certain layers. Hence we propose a more general solution: use 1×1 convolution to exchange the information of different groups. And we call this method "gcos": *Group COnvolution with Shuffling*.

Fig. 4 shows the framework of gcos. We divide each convolution layer into two layer. The first layer is a normal group convolution layer with g groups. We require the number of filters be a multiplier of g, and groups are formed by contiguous channels (e.g., channels with indexes $1, 2, \ldots, K/g$ form the first group). Channels in the ith output group are connected only to channels in the ith input group. Hence, the number of parameters is reduced to $1/g$ with the risk of large accuracy drop. Then, we use 1×1 convolution to exchange group information. In the experiments section, we will show that these 1×1 convolution filters play a crucial role in gcos.

This idea is similar to relevant ones in MobileNet [28] and ShuffleNet [29]. In MobileNet, the 1×1 convolution is also adopted. Combined with depthwise convolution, they have achieved very high efficiency. However, depthwise convolution is not applicable in our framework. We find that the model accuracy is damaged greatly when $g > 8$. In other words, depthwise convolution works well in MobileNet, but fails in VGG16. Hence, gcos can be viewed as a generalization of depthwise convolution. ShuffleNet proposes the channel shuffle operation to address the "information blocking" problem stated above. However, this layer is not supported by any off-the-shelf deep learning library. By contrast, our 1×1 convolution is general and simple.

Yet there is still one unanswered question: how to initialize these 1×1 convolution? They may be initialized via methods like "Xavier". But this may lead to great performance change, and need more epochs during fine-tuning. Following the key idea of ThiNet, we advise that the best initialization method is using "minimizing activation reconstruction error". Again, some instances of input tensors (the activation of group convolution) and original activation tensors (like the output of top row in Fig. 4) will be collected. Since 1×1 convolution is in fact a regression problem, we can recover filter weights via least squares.

3.4 Pruning Strategy

In this part, we introduce our different processing strategy for different CNN structure. There are mainly two types of network architecture: the traditional convolutional/fully-connected architecture, and recent structural variants. The former is represented by AlexNet [1] or VGGNet [2], while the latter mainly includes some recent networks like GoogLeNet [30] and ResNet [3]. The main difference between these two types is that more recent networks usually replace the FC (fully-connected) layers with a global average pooling layer [17], [31], and adopt some novel network structures like Inception in GoogLeNet or residual blocks in ResNet.

We use different strategies to prune these two types of networks. For VGG-16, we notice that more than 90 percent FLOPs exist in the first 10 layers (conv1-1 to conv4-3), while the FC layers contribute nearly 86.41 percent parameters. Hence, we prune the first 10 layers for acceleration, but replace the FC layers with a global average pooling layer. Although the proposed method is also valid for FC layers, we believe removing them is simpler and more efficient.

For ResNet, there are some restrictions due to its special structure. For example, the channel number of each block in the same group needs to be consistent in order to finish the sum operation. Thus, it is hard to prune the last convolution layer of each residual block directly. To address this problem, [19] only pruned the first layer of each basic residual block while keeping the last layer fixed. We adopted a similar strategy to tackle the bottleneck residual block. As illustrated in Fig. 5, only the first two layers in each bottleneck block will be pruned, because these two layers occupy the most FLOPs. As for the Batch Normalization (BN) layer, since it is channel-wise independent, we remove the corresponding BN channel when a filter is discarded.

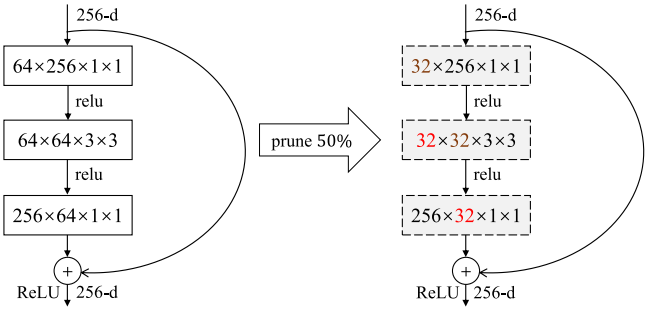


Fig. 5. Illustration of the ResNet pruning strategy. For each residual block, we only prune the first two convolution layers, keeping the block output dimension unchanged.

4 EXPERIMENTS

We empirically study the performance of ThiNet here. In the first part, we compare our filter level pruning strategy with several state-of-the-art methods. Experimental results show that our method is significantly better than others. For a fair comparison, gcos is included in this part. Two widely used networks are pruned on ILSCVR-12 [32]: VGG-16 [2] and ResNet-50 [3]. Then, we will add gcos into our framework. Two small models produced via our ThiNet are introduced in this part. Finally, we present several applications of ThiNet. For random selection, we report the averaged results of 3 repeated experiments. The number of runs is 1 by default. All the experiments are conducted within Caffe [33].

4.1 Part 1 of ThiNet: Pruning

4.1.1 Different Filter Selection Criteria

Baseline Methods. There are some heuristic criteria to evaluate the importance of each filter in the literature. We compare our selection method with several (non-)data driven criteria to demonstrate the effectiveness of our evaluation criterion. These methods are briefly summarized as follows.

- *random*. Filters are randomly discarded.
- *weight sum* [19]. This criterion calculates absolute sum of each filter i as its importance score: $s_i = \sum |\mathcal{W}(i,:,:,:)|$.
- *APoZ (Average Percentage of Zeros)* [20]. The sparsity of each channel in output activations $\mathcal{I}$ is calculated as its importance score: $s_i = \frac{1}{N}\sum \text{sparsity}(\mathcal{I}(i,:,:))$, where N is the dataset size, $\text{sparsity}()$ calculates the percentage of zeros.
- *mean-mean* [19]. $s_i = \frac{1}{N}\sum \text{mean}(\mathcal{I}(i,:,:))$.
- *mean-std* [19]. $s_i = \frac{1}{N}\sum \text{std}(\mathcal{I}(i,:,:))$.
- *mean-ℓ_1* [19]. $s_i = \frac{1}{N}\sum ||\mathcal{I}(i,:,:)||_1$.
- *mean-ℓ_2* [19]. $s_i = \frac{1}{N}\sum ||\mathcal{I}(i,:,:)||_2$.
- *var-ℓ_2* [34]. $s_i = \frac{1}{N}\sum \text{var}(||\mathcal{I}(i,:,:)||_2)$.

Except for APoZ, all these methods treat filters with higher scores as more important, which is motivated by the intuition that an unimportant feature map has similar or small outputs. As for APoZ, a higher score means almost all of output elements are zeros, which should be removed.

Datasets. To compare these different selection methods, iterative pruning is evaluated on two different datasets:

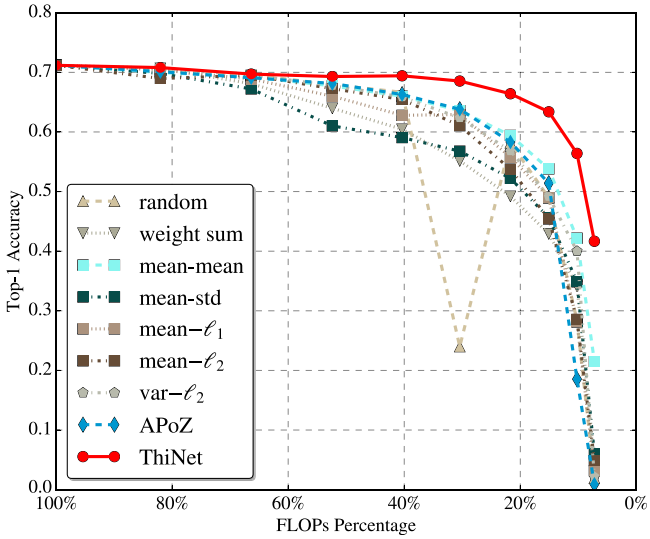


Fig. 6. Performance comparison of different channel selection methods: Pruning the VGG-16-GAP model on CUB-200 with different compression rates. For random selection, we run it 3 times and report the mean value. (This figure is best viewed in color and zoomed in.)

- *CUB-200* [35]: This is a typical dataset for fine-grained classification, which aims at recognizing bird subcategories. This dataset contains 11,788 images of 200 different bird species (5994/5794 images for training/test, respectively). Except for labels, no additional supervised information (e.g., bounding box) is used.
- *Indoor-67* [36]: This dataset contains 67 categories for indoor scene recognition, which is a challenging problem. We follow the official train/test split (5360 training and 1340 test images).

Many existing model compression algorithms reported their results on a small dataset like MNIST [37] or CIFAR-10 [38]. However, these datasets are relatively simple. Different algorithms often generate very similar results with negligible differences. Hence, we think that comparing on a tough but small dataset is necessary. By contrast, recognition on these two datasets are very challenging due to the low inter-class but high intra-class variations.

Implementation Details. Following the pruning strategy in Section 3.4, all the FC layers in VGG-16 are removed, replaced by a global average pooling layer. We then fine-tune this model for 21 epochs using SGD. Weight decay is set to 0.0005, momentum is 0.9 and batch size is set to 32. The initial learning rate starts from 0.001, and is divided by 10 in every 7 epochs. Starting from this fine-tuned model (VGG-16-GAP), we then prune the network layer by layer with different compression rates. For data-driven methods, we extracted features on all training images. And, these intermediate representations need to be re-generated at each layer. After pruning, we fine-tune the model by one epoch, and 12 epochs in the final layer to improve accuracy. This procedure is repeated several times with different channel selection strategies. For a fair comparison, all the parameters are kept the same: using SGD with 32 batch size, 10^{-4} learning rate, 0.9 momentum and 0.0005 weight decay.

Overall Performance Comparison. Figs. 6 and 7 show the pruning results on CUB-200 and Indoor-67, respectively. ThiNet achieves higher accuracy compared with other selection methods, especially when we discard more filters.

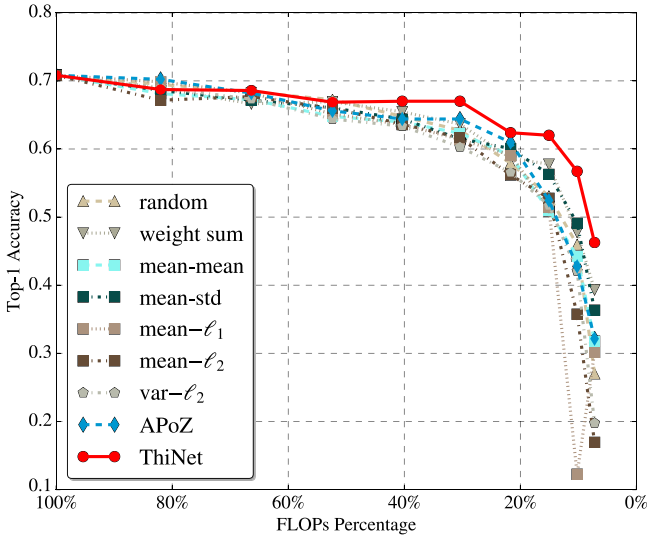


Fig. 7. Performance comparison of different channel selection methods: Pruning the VGG-16-GAP model on Indoor-67 with different compression rates. For random selection, we run it 3 times and report the mean value. (This figure is best viewed in color and zoomed in.)

One interesting observation is: random selection shows a pretty good result, even better than heuristic criteria in some cases. In fact, according to the property of distributed representations (i.e., each concept is represented by many neurons; and, each neuron participates in the representation of many concepts [39], [40]), randomly selected channels may be quite powerful in theory. However, this criterion is not robust. As shown in Fig. 6, it can lead to very bad result and the accuracy is very low after all layers are compressed. Thus, random selection is not applicable in practice.

Weight sum has pretty poor accuracy on CUB-200, but shows good performance on Indoor-67, which means weight sum is not very robust for different datasets. This result is reasonable, since it only takes the magnitude of kernel weights into consideration, which is not directly related to the final classification accuracy. Small weights may still have large impact. When we discard a large number of filters with small weights at the same time, the final accuracy can reduce a lot. For example, in the CUB-200 experiment, if we removed 60 percent filters in conv1-1 using the weight sum criterion, the top-1 accuracy is only 40.99 percent (before fine-tuning), while the random criterion is 51.26 percent. In contrast, our method reaches 70.80 percent. The accuracy loss of weight sum is so large that fine-tuning cannot completely recover it from the drop.

Layer-Wise Comparison and Running Speed. Table 1 shows the layer-wise pruning results on CUB-200 when the compression ratio r is set to 0.7. ThiNet achieves the best performance among these baselines. We also report the running speed of different channel selection methods. For data-driven methods, its running speed depends on dataset size and model inference speed. The VGG-16-GAP model takes 216s on a K80 GPU to extract features from all 5994 training images. Our greedy selection algorithm takes only 49.3s to select the important filters and corresponding scaling vector, which is even faster than some heuristic methods.

4.1.2 Pruning VGG-16 on ImageNet

We now evaluate the performance of the proposed ThiNet method on the large-scale ImageNet classification task.

Implementation Details. The ILSCVR-12 dataset [32] consists of over one million training images drawn from 1000

TABLE 1
Layer-Wise Pruning Results Using Different Channel Selection Criteria

	conv1-1	conv1-2	conv2-1	conv2-2	conv3-1	conv3-2	conv3-3	conv4-1	conv4-2	conv4-3
random	0.635 (0.0639ms) 0.701	0.576 0.688	0.547 0.680	0.510 0.672	0.615 0.668	0.532 0.652	0.405 0.647	0.539 0.636	0.510 0.622	0.493 0.679
weight sum	0.678 (1.4s) 0.699	0.658 0.696	0.611 0.685	0.311 0.622	0.376 0.590	0.370 0.566	0.256 0.557	0.237 0.541	0.328 0.533	0.426 0.640
mean-mean	0.650 (236.1s) 0.701	0.635 0.686	0.571 0.682	0.351 0.672	0.592 0.666	0.580 0.649	0.534 0.645	0.572 0.631	0.557 0.616	0.449 0.677
mean-std	0.629 (262.6s) 0.697	0.583 0.678	0.419 0.645	0.336 0.589	0.444 0.566	0.349 0.565	0.315 0.545	0.278 0.521	0.343 0.515	0.303 0.610
mean-ℓ_1	0.614 (240.1s) 0.692	0.553 0.678	0.613 0.662	0.448 0.636	0.560 0.627	0.553 0.618	0.411 0.610	0.457 0.596	0.516 0.583	0.385 0.660
mean-ℓ_2	0.681 (2483.7s) 0.706	0.596 0.675	0.647 0.670	0.400 0.651	0.564 0.642	0.590 0.637	0.426 0.628	0.515 0.610	0.523 0.603	0.448 0.673
var-ℓ_2	0.705 (248.5s) 0.707	0.627 0.694	0.671 0.694	0.583 0.673	0.582 0.663	0.621 0.649	0.545 0.648	0.556 0.629	0.543 0.620	0.459 0.678
APoZ	0.675 (385.7s) 0.706	0.593 0.697	0.480 0.689	0.613 0.685	0.577 0.674	0.563 0.665	0.532 0.654	0.601 0.647	0.539 0.627	0.484 0.681
ThiNet	**0.713** (265.3s) **0.711**	**0.708** **0.709**	**0.709** **0.707**	**0.684** **0.699**	**0.687** **0.690**	**0.681** **0.690**	**0.674** **0.680**	**0.660** **0.676**	**0.652** **0.666**	**0.647** **0.693**

The first/second row of each method means accuracy before/after fine-tuning. These results are conducted on K80 GPU to prune the VGG-16-GAP model on CUB-200 with compression ratio $r = 0.7$, i.e., 70 percent filters are preserved after pruning. For conv1-1, the running time of selecting unimportant filters is also reported (model forward time is around 216s).

categories. We randomly select 10 images from each category in the training set to comprise our evaluation set (i.e., collected training examples for channel selection). And for each input image, 10 instances are randomly sampled from different channels and spatial locations as described in Section 3.2.4. Hence, there are in total 100,000 training samples used for finding the optimal channel subset via Algorithm 1. Finally, top-1 and top-5 classification performance are reported on the 50k standard validation set, using the single-view testing approach (central patch only).

During fine-tuning, images are resized to 256×256, then 224×224 random cropping is adopted to feed the data into network. Horizontal flip is also used for data augmentation. At the inference stage, we center crop the resized images to 224×224. No other tricks are used here. The whole network is pruned layer by layer and fine-tuned in one epoch with 10^{-3} learning rate. We find the last layer of each group (i.e., conv1-2, conv2-2, conv3-3) is more sensitive to pruning. For example, pruning conv1-1 and conv2-1 only bring 2.6 and 2.2 percent top-1 accuracy drop, respectively. However, when we prune conv1-2 and conv2-2, the top-1 accuracy is 4 and 11 percent lower than the model before pruning, respectively. Similar observation is also found for ResNet-34 [19]. Hence, we fine-tune these layers with one additional epoch of 10^{-4} learning rate to prevent large accuracy drop. When pruning the last layer, more epochs (12 epochs) are adopted to get an accurate result with learning rate varying from 10^{-3} to 10^{-5}. We use SGD with mini-batch size of 64, and other parameters are kept the same as the original VGG paper [2].

Training Examples. First, we want to study the impact of different training examples for channel selection. Our sample number m is determined by the image number in each category and the location number in each image. Hence, we change these two variables with 5 choices: 1, 5, 10, 15, 20. We find that there is no obvious difference among these settings. For example, if we randomly sample 1 image per category and 1 location per image, the accuracy of pruning the conv1-1 layer is 70.48 percent without fine-tuning. However, when we increase both values to 20, its accuracy is 70.41 percent. Our selection method is very robust to the choice of training examples. The standard deviation of these different settings is only 0.21 percent. Hence, we use the setting of 10 images and 10 locations. In this setup, the collected intermediate representation takes up 51.2 MB disk space when the channel number is 64 and 409.6 MB with 512 channels.

Compression Ratio. In some real-world application scenarios, the inference speed is constrained. For example, a scene segmentation network should return predictions within 50 ms in self-driving vehicles for safety considerations. Hence, the FLOPs of this network should be less than a threshold. And we can calculate the corresponding compression ratio according to this threshold value. In this section, we set the compression ratio to 0.5 and 0.4 in order to compare it with previous methods. For simplicity, all the layers are pruned using a fixed compression ratio. It is difficult to find a criterion to obtain the best pruning ratios for different layers.

Compressed ThiNet Models. We compare the performance of our ThiNet approach with other state-of-the-art filter level pruning methods in Table 2. We adopt two different strategies to prune VGG16, and obtain two types of ThiNet models:

TABLE 2
Comparison Among Several State-of-the-Art Pruning Methods on VGG-16 and ILSVRC-12

Method	Top-1 Acc.	Top-5 Acc.	#Param.	#FLOPs
VGG-16 [2][1]	71.50%	90.01%	138.34 M	30.94 B
connection pruning [9]	68.66%	89.12%	10.30 M	6.50 B
APoZ-1 [20]	66.20%	87.60%	67.81 M	–
APoZ-2 [20]	70.17%	89.69%	51.24 M	–
Taylor-1 [21]	–	87.00%	–	11.50 B
Taylor-2 [21]	–	84.50%	–	8.00 B
weight sum [19][2]	69.35%	89.13%	131.44 M	9.58 B
Channel-Pruning (5×) [41][3]	67.80%	88.10%	130.87 M	7.03 B
Train from scratch[4]	67.00%	87.45%	131.44 M	9.58 B
ThiNet-Conv-1	69.74%	89.41%	131.44 M	9.58 B
ThiNet-Conv-2	69.11%	88.86%	130.50 M	6.91 B
ThiNet-GAP	67.69%	88.13%	8.32 M	9.34 B

[1] *In some papers (e.g. [9], [20]), VGG-16 accuracy (68.34/88.44 percent) was tested on resized 256×256 images. But, this model was not trained using this resolution. Here, we use the result reported in MatConvNet: http://www.vlfeat.org/matconvnet/pretrained/.*
[2] *The original paper did not report their accuracy on VGG-16. We reimplement it, and report the results using our pruning framework.*
[3] *https://github.com/yihui-he/channel-pruning/releases/tag/channel_pruning_5x.*
[4] *We train the ThiNet-Conv-1 model from scratch.*
Here, M/B means million/billion (10^6/10^9), respectively.

- *ThiNet-Conv*: This model only prune the first 10 convolution layers and keep the FC layers. We use ThiNet-Conv-1 and ThiNet-Conv-2 to denote the models pruned with two different compression ratio 0.5 and 0.4 (i.e., only 50 or 40 percent filters are preserved in each layer till conv4-3).
- *ThiNet-GAP*: Based on the ThiNet-Conv-1 model, we replace FC layers with a GAP (global average pooling) layer and fine-tune this model for 12 epochs with the same hyper-parameters. The classification accuracy of GAP model is slightly lower than the original model, but the model size has been reduced dramatically.

Pruning versus Training from Scratch. First, we want to answer the following interesting question: why do we need model pruning rather than training it from scratch? In order to answer this question, we train a model from scratch with the same architecture as ThiNet-Conv-1. Unfortunately, the top-1/top-5 accuracy are only 67.00/87.45 percent, which are much worse than our pruned network. Similar observation on CIFAR-10 is also made in previous work such as [19]. Now, we demonstrate that this conclusion can be extended to the large scale ImageNet recognition task. In fact, deep learning is a highly non-convex optimization problem, a certain degree of parameter redundancy seems *necessary* during *model training*, especially for a complex model. Hence, we need to remove these redundant parameters *after training*.

ThiNet versus Connection Pruning. Connection pruning is a classic CNN pruning method proposed by Han et al. [9]. Although it achieves impressive accuracy, the non-structured sparse model is hard to harvest actual computational savings [10] due to the irregular convolution. As shown in [16], filter level pruning is more challenging, causing nearly 3 percent lower top-5 accuracy than irregular connection pruning with the same sparsity. However, ThiNet can obtain a comparable accuracy as connection pruning.

TABLE 3
Performance of Pruning ResNet-50 on ImageNet Using Different Compression Rates

Model	Top-1	Top-5	#Param.	#FLOPs	Speed (images/s)
Original[1]	75.30%	92.20%	25.56 M	7.72 B	295.12
ThiNet-70	74.03%	92.11%	16.94 M	4.88 B	334.87
ThiNet-50	72.03%	90.99%	12.38 M	3.41 B	373.92
ThiNet-30	68.17%	88.86%	8.66 M	2.20 B	397.86

1 The accuracy of ResNet is tested using official 1-crop validation setting: center 224 × 224 crop from resized image with shorter side=256 (https://github.com/KaimingHe/deep-residual-networks).
M/B means million/billion, respectively. Inference speed is tested on one GTX 1080 GPU with batch size 32. Here, speed $= 1000 \div (millisecond/iteration) * (batchsize/iteration)$.

ThiNet versus other Filter Pruning Methods. Next, we compare the proposed ThiNet with other filter level pruning methods. APoZ [20] aims at reducing parameter numbers, but only prunes last few convolution and FC layers. Such a strategy has little effect on model acceleration, since almost all FLOPs are spent on convolution layers. As shown in Section 4.1.1, ThiNet is better than APoZ. As for Taylor methods [21], they obtain much worse results than ThiNet. We reimplemented weight sum using our ThiNet pipeline since the results on VGG16 are not reported. Note that different fine-tuning framework may lead to very different results. Hence, the accuracy may be different if Li et al. [19] had done this using their own framework. Because the rest setups are the same, it is fair to compare weight sum with ThiNet, and ThiNet has obtained better results. Finally, we compare ThiNet with Channel-Pruning [41], which is very similar to ours with the same design idea: minimizing reconstruction error. The major difference is the selection strategy: they use LASSO regression while ours is a greedy based method. As shown in Table 2, ThiNet is much better than Channel-Pruning with even more FLOPs reduction.

4.1.3 ResNet-50 on ImageNet

We also explore the performance of ThiNet on the compact CNN architecture: ResNet [3]. We select ResNet-50 as a representative of the ResNet family, which has exactly the same architecture and little difference with others.

Implementation Details. Similar to VGG-16, we prune ResNet-50 from block 2a to 5c iteratively. In addition to filters, the corresponding channels in batch normalization layer are also discarded. After pruning, the model is fine-tuned in one epoch with fixed learning rate 10^{-4}. And 9 epochs fine-tuning with learning rate changing from 10^{-3} to 10^{-5} is performed at the last round. Other parameters are kept the same as those in our VGG-16 pruning experiment.

Performance on ResNet-50. Table 3 shows the results of ThiNet. We prune ResNet-50 with 3 different compression rates (preserve 70, 50, 30 percent filters in each block, respectively). Unlike VGG-16, ResNet is more compact. There exists less redundancy, thus pruning a large amount of filters seems to be more challenging. In spite of this, our model ThiNet-50 can still obtain $2.26\times$ acceleration (FLOPs reduction) with 1.21 percent top-5 accuracy drop. Further pruning can also be carried out, leading to a much smaller model at the cost of more accuracy drop.

We also report the inference speed of these pruned models in Table 3. To be consistent with our speed test settings (in Table 4, we test model speed on mobile phone using the caffe2 libraries), each caffe model is first converted into a caffe2 model using the official scripts. We then test the inference speed of these four models with batch size 32.

In this experiment, we only prune the first two layers of each block in ResNet for simplicity, leaving the block output and projection shortcuts unchanged. Pruning these parts would lead to further compression, but can be quite difficult, if not entirely impossible. And this exploration seems to be a promising extension for the future work.

4.2 Part 2 of ThiNet: gcos

To explore the limits of small models that can be supported by off-the-shelf deep learning libraries (i.e., we do not want to change the original network structures), we can further reduce model size via the proposed gcos method.

Small Models. As discussed in Section 3.3, we first use normal group convolution to reduce model parameters, followed by 1×1 convolution to solve the information blocking problem. This processing is also performed layer by layer with the same fine-tuning hyper-parameters as in Section 4.1.2. We get two small models using our ThiNet pipeline. The settings are summarized as follows.

- *ThiNet-Small (4.67MB):* Given the original VGG-16 model, we prune it from conv1-1 to conv5-3 with 0.5 compression rate (i.e., only half of the filters are preserved) and replace the FC layers with global average pooling as we have done in Section 4.1.2. Then, we use gcos to further reduce model size. We divide

TABLE 4
ImageNet Performance of Small ThiNet Models Generated from VGG16

	Accuracy		Model Information			Inference Speed (images/s)		
Model	Top-1	Top-5	#Param.	#FLOPs	model size	Nvidia GTX 1080 Ti	Intel Core i7-7700	iPhone 6S
ThiNet-Tiny	57.41%	80.52%	694.96 K	1.16 B	2.66 MB	1406.59	20.35	8.47
ThiNet-Small	62.97%	85.06%	1.22 M	2.62 B	4.67 MB	678.33	10.24	3.71
SqueezeNet [42]	57.67%	80.39%	1.24 M	1.72 B	4.76 MB	1201.20	29.32	8.62
SqueezeNet-DSD [43]	61.80%	83.50%	1.24 M	1.72 B	4.76 MB	1201.20	29.32	8.62
AlexNet [1]	57.28%	80.27%	60.95 M	1.45 B	232.57 MB	2745.60	23.90	5.26[1]

1 In caffe2, the default protobuf limit is only 64 MB, which is much smaller than AlexNet. In order to load AlexNet, we increase the limit to 300 MB. However, larger protobuf limit will slightly slow down inference speed.
We also show the results of other small CNN models. Inference speed is tested with batch size 64, except for mobile phone which is tested using single image. Here, $speed = 1000 \div (millisecond/iteration) * (batchsize/iteration)$.

the first few layers (from conv1-2 to conv3-3) into 4 groups, but divide the rest layers into 8 groups due to the larger memory consumption. Each group convolution layer is followed by 1×1 convolution. We do not change conv1-1, because its input has only 3 channels (the RGB image).

- *ThiNet-Tiny (2.66 MB):* This processing pipeline is similar to ThiNet-Small with only two differences. First, we prune it from conv1-1 to conv5-2 with 0.25 compression rate to get a smaller model. As for conv5-3, which is directly related to the final feature representation, we only prune half of the filters. Second, we divide the last few layers (from conv3-1 to conv5-3) into 4 groups.

Results. Table 4 shows the performance of these two small models. ThiNet-Tiny is a very small model, which takes only 2.66 MB disk space (1MB = 2^{20} bytes) but still has AlexNet-level accuracy. In contrast, the recently proposed compact network, SqueezeNet [42], has $1.78\times$ parameters and $1.48\times$ FLOPs. Similarly, AlexNet is much larger, it takes more than 232 MB space and needs $1.25\times$ FLOPs during inference compared with ThiNet-Tiny. These three models have the same level of accuracy, but ThiNet-Tiny shows higher computational efficiency.

Han et al. [43] proposed a dense-sparse-dense (DSD) training flow to regularize deep neural networks, and achieved better performance on SqueezeNet. This model is denoted by "SqueezeNet-DSD". It has exactly the same structure as the original SqueezeNet. The only difference exists in its training flow. Here, we compare this model with ThiNet-Small. ThiNet-Small achieves higher classification accuracy even with less parameters than SqueezeNet-DSD.

Actual Running Speed. Finally, we report the inference speed of each model in Table 4, which is tested using caffe2. We run these 5 models on 3 different devices: GPU, CPU and mobile phone. As we can see, AlexNet has the fastest running speed on GPU, but will slow down when applied on CPU or mobile phone. It may be that caffe2 does not support group convolution well on CPU or embedded devices. The same phenomenon can also be observed on ThiNet-Tiny, which shows faster speed on GPU, but slightly slower on CPU and mobile phone compared with SqueezeNet. SqueezeNet shows pretty good performance on different devices. In contrast, ThiNet-Tiny achieves comparable speed-up as SqueezeNet with fewer parameters. Note that, SqueezeNet adopts a special structure, namely the "Fire module", which is parameter efficient but relies on manual network structure design. However, ThiNet is a unified framework, and higher accuracy can be obtained if we start from a more accurate model. In fact, this is a tradeoff among inference speed, model size and manual network design effort. As shown in Figs. 6 and 7, this tradeoff can be realized via different compression ratio. In real applications, the speed and model size constraints can vary significantly, and our pruning method provides a flexible solution.

4.3 Applications

We then study some real applications of our pruned model. In this section, we will compare ThiNet with SqueezeNet and AlexNet in several different computer vision tasks, including classification, image retrieval, object detection, semantic segmentation and style transfer. We show that our pruned ThiNet models achieve impressive performance on these tasks.

TABLE 5
Image Classification on 5 Benchmark Datasets Using Different Networks

Model	Size	Caltech101	CUB200	Indoor67	Action40	Event8
AlexNet	232.57 MB	92.00%	57.28%	59.55%	62.00%	93.39%
SqueezeNet	4.76 MB	90.74%	65.34%	62.99%	65.76%	93.69%
ThiNet-Tiny	2.66 MB	90.07%	64.67%	62.69%	67.32%	94.28%
SqueezeNet-DSD	4.76 MB	90.28%	66.97%	63.81%	68.08%	93.10%
ThiNet-Small	4.67 MB	91.68%	68.33%	67.01%	72.07%	94.48%

Our two pruned ThiNet models achieve impressive performance, showing powerful generalization ability.

4.3.1 Image Classification

We first consider a more practical scenario: image recognition with small models on some domain-specific datasets. This is a very common requirement in real-world applications, since we will not directly apply ImageNet model in a real application. The comparison is performed on 5 popular benchmark datasets, which are summarized as follows:

- *Caltech101 [44]:* A typical dataset for *object recognition*. We randomly select 30 images per category to constitute the training set, and regard the rest of images as test set. Hence, there are total 5994/5794 images for training/test respectively.
- *CUB200 [35]:* A popular *fine-grained* dataset, which has been introduced in Section 4.1.1.
- *Indoor67 [36]:* A popular *indoor scene* dataset, which has been introduced in Section 4.1.1.
- *Action40 [45]:* The Stanford 40 *Action* dataset contains images of humans performing 40 actions. We follow the official suggested train/test split to organize this dataset, which contains 4000/5532 images.
- *Event8 [46]:* The UIUC Event8 dataset contains 8 *sports event* categories. We randomly select 70 images per category (560 images) as the training set, and take the rest (1014 images) as the test set.

For each dataset, the images are first resized into 224×224. Then each model is fine-tuned using the same hyper-parameters. We use SGD to train model, and change learning rate from 10^{-3} to 10^{-6}.

Table 5 shows the comparison results (top-1 accuracy) on these 5 benchmarks. As expected, due to the structure constraint, AlexNet has very limited generalization ability. Except for Caltech101, AlexNet is worse than ThiNet-Tiny even with $87\times$ more parameters. As for SqueezeNet and ThiNet-Tiny, although these two models have similar performance on ImageNet, ThiNet-Tiny can outperform SqueezeNet on some domain-specific datasets (Action40 and Event8), and shows comparable results on others. Since our tiny model only takes up 2.66 MB space (almost half the size of SqueezeNet), we think it is acceptable with some negligible accuracy drop. And we believe this model is small enough to be loaded in any small device (mobile phones or embedded gadgets).

We then compare the performance of two more accurate models: SqueezeNet-DSD and ThiNet-Small. Both of them have similar model size and accuracy on ImageNet according to Table 4. However, ThiNet-Small presents much stronger generalization ability when transferring to other

TABLE 6
Fine-Grained Image Retrieval Using the SCDA Method [47]

Model	Size	top-1	top-5
AlexNet	232.57 MB	65.09	71.09
SqueezeNet	4.76 MB	71.87	77.47
ThiNet-Tiny	2.66 MB	72.72	78.03
SqueezeNet-DSD	4.76 MB	76.42	80.88
ThiNet-Small	4.67 MB	79.07	83.31

Performance is evaluated by top-1/ top-5 mAP values (higher is better).

domain-specific tasks. As shown in Table 5, ThiNet-Small outperforms SqueezeNet-DSD on all five datasets, and in some cases by a large margin (Indoor67 and Action40).

4.3.2 Image Retrieval

We have shown that our pruned models (ThiNet-Small and ThiNet-Tiny) present impressive performance on image recognition tasks. Now, we are interested in other computer vision tasks to see how explicit improvement could be achieved within the ThiNet models.

First, we will explore the performance of fine-grained image retrieval using the SCDA method [47], which aims to query the same species images (e.g., birds, flowers or dogs) in the pure unsupervised setting.[1] We replace the base feature extractor with our 5 models (without fine-tuning), and use the base SCDA setting in [47] to generate useful deep descriptors. The base setting means only the last convolution features are used in our experiment without any other tricks (such as image flipping or aggregating earlier layer) for simplicity. Then, the extracted features are pooled with global max/average pooling, and concatenated to form the final SCDA feature.

We compare different base models on one popular fine-grained benchmark: Oxford-IIIT Pets [48]. This dataset consists of 37 different category of pets (cats or dogs) with roughly 200 images for each class. We follow the official suggestion to split the training/test set (3680/3669 images). Only raw images are used, excluding any supervision information. We use the test images to query the training set, and report retrieval performance using mAP values. Since fine-grained retrieval is a novel and challenging task, only top-1 and top-5 mAP values are reported here.

Table 6 shows the retrieval results using different base models. Again, AlexNet has the worst mAP values because of the limited feature generalization ability. As for SqueezeNet and ThiNet, our two ThiNet models present consistently better retrieval performance than SqueezeNet with 0.85/2.65 higher top-1 mAP values. This is reasonable, because image retrieval is closely related to feature generalization ability (SCDA is an unsupervised method). Our ThiNet achieves stronger generalization on the classification task, thus have a high probability of showing better retrieval results.

4.3.3 Object Detection

From now on, we study whether ThiNet can generalize well to other high level vision tasks. In this section, we study its object detection performance. To construct different detectors, we followed the strategy proposed in [49], which means we append the same blocks (conv-$\{6, 7, 8, 9\}$) on the top of different pretrained models. All the models are trained on the trainval set of VOC2007 and VOC2012, then tested on VOC2007 test images. We set the batch size to 8 in training, and fine-tune the model with 10^{-3} learning rate for the first 120k iterations, then 10^{-4} and 10^{-5} for next two 30k iterations, respectively. We use SGD to optimize the training process where the weight decay is set to 0.0005, momentum is 0.9 and gamma is 0.1.

TABLE 7
Object Detection Comparison on the VOC2007 Test Dataset [50] Using the SSD Method [49]

Model	Size	FPS	mAP
AlexNet	21.3 MB	93	51.7
SqueezeNet	17.1 MB	68	59.1
ThiNet-Tiny	13.5 MB	69	55.0
SqueezeNet-DSD	17.1 MB	68	43.9
ThiNet-Small	16.1 MB	45	66.4
SSD300 [49]	105.2 MB	22	77.2
Fast-YOLO [51]	180 MB	89	52.7
Tiny-YOLO [52]	63.5 MB	66	57.1

Speed is tested on one Nvidia Tesla K80 GPU card (images/s).

TABLE 8
Semantic Segmentation on VOC2011 [50]

Model	Size	FPS	MeanIU
AlexNet	223.9 MB	15.86	49.53
SqueezeNet	6 MB	22.76	42.7
ThiNet-Tiny	4.2 MB	22.52	46.43
SqueezeNet-DSD	6 MB	22.77	39.56
ThiNet-Small	6.2 MB	16.66	51.39

Speed (images/s) is tested on one Nvidia Tesla K80 GPU card with 500×500 *resized images.*

Experimental results are reported in Table 7. Interestingly, SqueezeNet shows pretty good result, while SqueezeNet-DSD fails in our experiment with 15.2 lower mAP. The only difference between these two models is the training method. It seems that the DSD approach may hurt a model's generalization performance. The mAP of ThiNet-Tiny is slightly lower than SqueezeNet, but still higher than AlexNet. ThiNet-Small shows the best performance among these five models at the cost of slightly lower inference speed.

We also provide 3 popular detection benchmarks for a rough comparison, namely SSD300 [49], Fast-YOLO [51] and Tiny-YOLO [52]. As we can see, the performance of ThiNet-Small is much higher than two YOLO models, but lower than SSD300. However, the model size of ThiNet-Small is $3.9 \sim 11.2\times$ smaller than these three models. And its inference speed is also $2\times$ faster than SSD300.

4.3.4 Semantic Segmentation

Semantic segmentation is another high level vision task we use to test ThiNet. We use the fully convolutional networks (FCNs) [53] as our segmentation method. Atrous spatial pyramid pooling (ASPP) [54], which aims to capture objects and image context at multiply scales, is also adopted to further improve the performance. The official code of FCNs is used in our experiment.[2] We fine-tune the models on the VOC2011

1. http://lamda.nju.edu.cn/code_SCDA.ashx

2. https://github.com/shelhamer/fcn.berkeleyvision.org

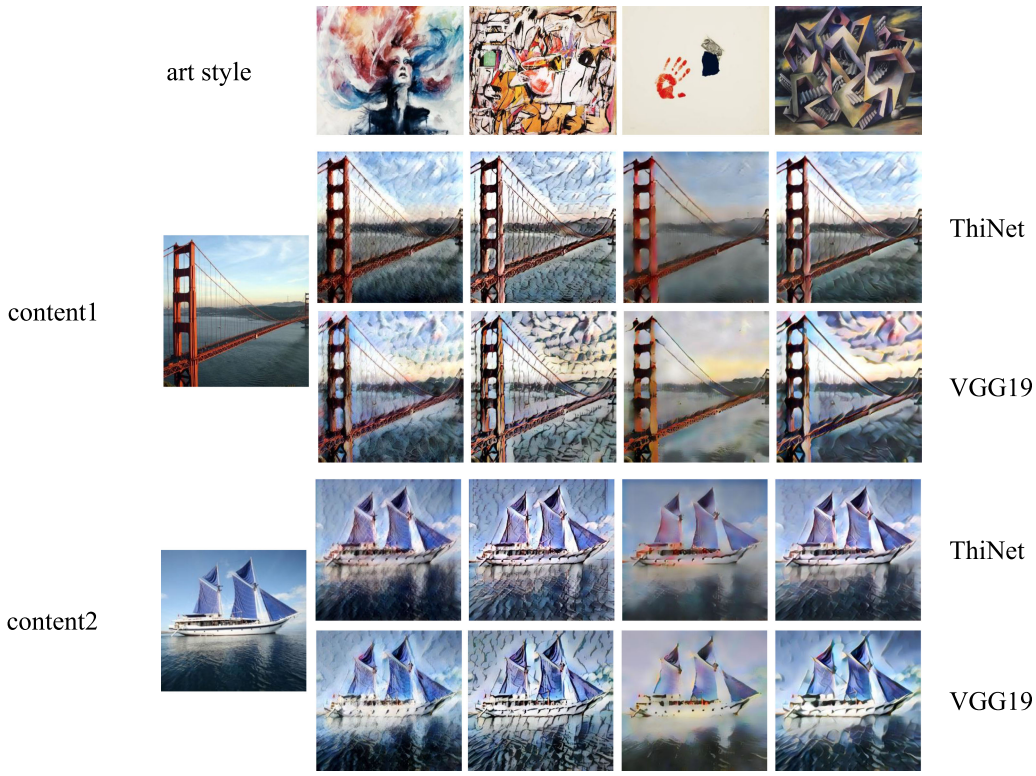


Fig. 8. Example style transfer results using different base models. There is no obvious winner between our ThiNet model and VGG19 according to the transfer results, but ThiNet model is $1.5/2.5\times$ faster in GPU/CPU. (This figure is best viewed in color and zoomed in.)

dataset [50]. All the images are resized to 500×500. For AlexNet, we follow the same setting of FCNs except for adding the ASPP block. As for ThiNet and SqueezeNet, we remove the global average pooling and fully-connected layer first, then add a ASPP block. All the training hyperparameters are kept the same as default FCNs codes. More details about the experimental settings can be referred to the FCNs paper.

Table 8 shows the segmentation results. One interesting observation is that the SqueezeNet-DSD model fails again, much worse than the original one. This phenomenon indicates the weakness of the DSD training method [43]: it may force the network to pay more attention to the classification task, which is harmful for model generalizing. In contrast, our ThiNet shows consistently better performance. ThiNet-Tiny is 3.73 percent higher than SqueezeNet with even smaller model size. And ThiNet-Small even outperforms AlexNet with 1.86 percent meanIU values as well as faster speed! This is a reasonable phenomenon. In AlexNet, the fully-connected layers are replaced by convolution layers. For a 500×500 input image, these replaced layers are still computation intensive. On the other hand, due to structure issues, the ASPP and upsampling operations are more time-consuming compared with ThiNet models. Hence, ThiNet-Small achieved higher performance with fewer parameters and faster inference speed than AlexNet.

4.3.5 Style Transfer

Finally, we would like to see how the proposed ThiNet could accelerate a more visually attractive vision task: style transfer. We use a recently proposed arbitrary style transfer algorithm AdaIN-Style [6] in our experiment. Since the VGG19 network [2] is more effective in style transfer, we pruned a VGG19 network first. The experiment setting is kept the same as what we presented in Section 4.1.2. We start from conv1-1, prune layer by layer using our ThiNet scheme with 0.5 compression rate, and finally stop at the conv5-1 layer.

Then the pruned VGG19 model is compared with the original VGG19 using the official AdaIN-Style codes.[3] For fairness of comparison, both models are trained with default hyperparameters. After that, we use the trained model to generate several transfer examples (all the tested content and style images are never used during training), and compare the different inference speed.

Fig. 8 shows some transfer results. We randomly select 2 content images with 4 different art styles. As we can see, there are no obvious winner between our ThiNet model and the original VGG19 network. Since art is one of the most subjective matters, it is hard to say which is better, though ThiNet results may lose some details slightly like color brightness. Overall, the compressed network can still generate acceptable images with different art styles.

We then test the actual acceleration ratio of our pruned model using the default 12 contents and 21 style images, hence there are in total 252 transfered results. The speed is calculated by the averaged inference time (exclude model loading cost). Table 9 shows the comparison results. On GPU, our ThiNet model can bring $1.5\times$ acceleration compared with the original one. On CPU, the speed-up ratio is improved to $2.5\times$ with a single thread. As for model size, the ThiNet model only takes up 14 MB disk space, while VGG model is 56.2 MB here. In a nutshell, our ThiNet method could do indeed aid in reducing computational time and storage footprint.

3. https://github.com/xunhuang1995/AdaIN-style

TABLE 9
Style Transfer Inference Speed Comparison for 512×512 Images

GPU (Nvidia Tesla M40)		CPU (Intel Xeon E5-2680)	
ThiNet	VGG19	ThiNet	VGG19
0.200s/image	0.297s/image	5.920s/image	14.956s/image

Our ThiNet model achieves $1.5/2.5\times$ *acceleration on GPU/CPU with negligible image quality loss.*

5 Conclusions

In this paper, we proposed ThiNet, an effective channel-wise pruning method for deep model acceleration and compression. We showed that the proposed scheme can significantly improve model pruning performance over existing methods. Furthermore, our pruned model can be combined with any existing model compression methods. We proposed gcos, an efficient group convolution approach to further reduce model size. The effectiveness of ThiNet was evaluated in several challenging computer vision tasks. Experimental results demonstrated the excellent performance of ThiNet.

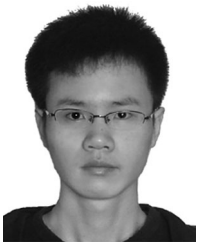
Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61772256 and No. 61422203, in part by Shanghai 'The Belt and Road' Young Scholar Exchange Grant No. (17510740100).

References

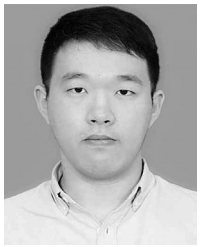
[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Advances Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–14.
[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[4] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1440–1448.
[5] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1520–1528.
[6] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 1501–1510.
[7] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Advances Neural Inf. Process. Syst.*, 1990, pp. 598–605.
[8] G. Chechik, I. Meilijson, and E. Ruppin, "Synaptic pruning in development: A computational account," *Neural Comput.*, vol. 10, no. 7, pp. 1759–1777, 1998.
[9] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Advances Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[10] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Advances Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.
[11] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 4820–4828.
[12] M. Denil, B. Shakibi, L. Dinh, and N. de Freitas, "Predicting parameters in deep learning," in *Proc. Advances Neural Inf. Process. Syst.*, 2013, pp. 2148–2156.
[13] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Advances Neural Inf. Process. Syst.*, 2014, pp. 1269–1277.
[14] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," in *arXiv:1207.0580*, pp. 1–18, 2012.
[15] V. Lebedev and V. Lempitsky, "Fast ConvNets using group-wise brain damage," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2554–2564.
[16] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. Dally, "Exploring the granularity of sparsity in convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2017, pp. 1927–1934.
[17] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proc. Advances Neural Inf. Process. Syst.*, 2016, pp. 2921–2929.
[18] R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 618–626.
[19] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–13.
[20] H. Hu, R. Peng, Y. W. Tai, and C. K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," in arXiv:1607.03250, pp. 1–9, 2016.
[21] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–17.
[22] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2736–2744.
[23] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," in *arXiv:1412.6115*, pp. 1–10, 2014.
[24] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2285–2294.
[25] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Representations*, 2016, pp. 1–14.
[26] V. Sindhwani, T. Sainath, and S. Kumar, "Structured transforms for small-footprint deep learning," in *Proc. Advances Neural Inf. Process. Syst.*, 2015, pp. 3088–3096.
[27] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 5058–5066.
[28] A. 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," in *arXiv:1704.04861*, pp. 1–9, 2017.
[29] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.
[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[31] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proc. Int. Conf. Learn Representations*, 2013, pp. 1–10.
[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F.-F. Li, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[33] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. ACM Int. Conf. Multimedia*, 2014, pp. 675–678.
[34] A. Polyak and L. Wolf, "Channel-level acceleration of deep face representations," *IEEE Access*, vol. 3, pp. 2163–2175, 2015.
[35] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD birds-200–2011 dataset," *California Institute of Technology, Pasadena, CA, Tech. Rep. CNS-TR-2011-001*, 2011.
[36] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 413–420.
[37] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[38] A. Krizhevsky, "Learning multiple layers of features from tiny images," MSc thesis, Department of Computer Science, University of Toronto, 2009.

[39] G. Hinton, "Learning distributed representations of concepts," in *Proc. Annu. Meeting Cognitive Sci. Society*, 1986, pp. 1–12.
[40] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.
[41] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 1389–1397.
[42] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and <0.5 MB model size," in *arXiv:1602.07360*, pp. 1–13, 2016.
[43] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, and B. Catanzaro, "DSD: Dense-sparse-dense training for deep neural networks," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–13.
[44] F.-F. Li, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories," *Comput. Vis. Image Understanding*, vol. 106, no. 1, pp. 59–70, 2007.
[45] B. Yao, X. Jiang, A. Khosla, A. Lin, L. Guibas, and F.-F. Li, "Human action recognition by learning bases of action attributes and parts," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2011, pp. 1331–1338.
[46] L.-J. Li and F.-F. Li, "What, where and who? Classifying events by scene and object recognition," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2011, pp. 1–8.
[47] X.-S. Wei, J.-H. Luo, J. Wu, and Z.-H. Zhou, "Selective convolutional descriptor aggregation for fine-grained image retrieval," *IEEE Trans. Image Process.*, vol. 26, no. 6, pp. 2868–2881, Jun. 2017.
[48] O.-M. Parkhi, A. Vedaldi, A. Zisserman, and C.-V. Jawahar, "Cats and dogs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3498–3505.
[49] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A.-C. Berg, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 21–37.
[50] M. Everingham, S.-A. Eslami, L. V. Gool, C.-K. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes challenge: A retrospective," *IEEE J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, Jan. 2015.
[51] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.
[52] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7263–7271.
[53] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, Apr. 2017.
[54] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A.-L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, Apr. 2018.

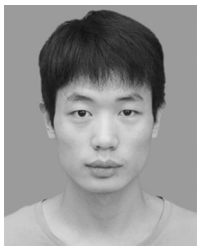


Jian-Hao Luo received the BS degree in the College of Computer Science and Technology from Jilin University, China, in 2015. He is currently working toward the PhD degree in the Department of Computer Science and Technology, Nanjing University, China. His research interests include computer vision and machine learning.

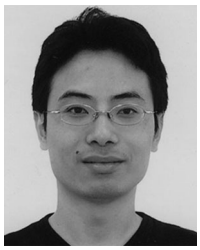
Hao Zhang received the BS degree from Nanjing University, China, in 2016. He is currently working toward the MS degree in the Department of Computer Science and Technology, Nanjing University, China. His research interests include computer vision and machine learning.



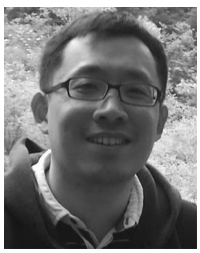
Hong-Yu Zhou received his BE and MS degrees from Wuhan University, China in 2015 and the Department of Computer Science and Technology, Nanjing University, China in 2018, respectively. His research interests include computer vision and machine learning.



Chen-Wei Xie received his BS and MS degree from Southeast University, China, in 2015 and the Department of Computer Science and Technology, Nanjing University, China in 2018, respectively. His research interests include computer vision and machine learning.



Jianxin Wu received the BS and MS degrees in computer science from Nanjing University, and the PhD degree in computer science from the Georgia Institute of Technology. He is currently a professor with the Department of Computer Science and Technology, Nanjing University, China, and is associated with the National Key Laboratory for Novel Software Technology, China. He has served as an area chair for CVPR, ICCV, and AAAI, and is an associate editor for the Pattern Recognition Journal. His research interests include computer vision and machine learning. He is a member of the IEEE.



Weiyao Lin received the BE and ME degrees from Shanghai Jiao Tong University, China, in 2003 and 2005, and the PhD degree from the University of Washington, Seattle, USA, in 2010. He is currently a professor at Department of Electronic Engineering, Shanghai Jiao Tong University, China. He has authored or coauthored more than 100 technical papers on top journals/conferences. His research interests include image/video processing, video surveillance, computer vision. He served as an associate editor for Journal of the *Visual Communication and Image Representation*, the *Signal Processing: Image Communication*, Circuits Systems and Signal Processing, and IEEE Access. He is a senior member of IEEE.