Centripetal SGD for Pruning Very Deep Convolutional Networks with Complicated Structure *

Xiaohan Ding 1  Guiguang Ding 1  Yuchen Guo 1  Jungong Han 2
1 Tsinghua University  2 Lancaster University
dxh17@mails.tsinghua.edu.cn dinggg@tsinghua.edu.cn \{yuchen.w.guo,jungonghan77\}@gmail.com

Abstract

The redundancy is widely recognized in Convolutional Neural Networks (CNNs), which enables to remove unimportant filters from convolutional layers so as to slim the network with acceptable performance drop. Inspired by the linearity of convolution, we seek to make some filters increasingly close and eventually identical for network slimming. To this end, we propose Centripetal 1 SGD (C-SGD), a novel optimization method, which can train several filters to collapse into a single point in the parameter hyperspace. When the training is completed, the removal of the identical filters can trim the network with NO performance loss, thus no finetuning is needed. By doing so, we have partly solved an open problem of constrained filter pruning on CNNs with complicated structure, where some layers must be pruned following others. Our experimental results on CIFAR-10 and ImageNet have justified the effectiveness of C-SGD-based filter pruning. Moreover, we have provided empirical evidences for the assumption that the redundancy in deep neural networks helps the convergence of training by showing that a redundant CNN trained using C-SGD outperforms a normally trained counterpart with the equivalent width.

1. Introduction

Convolutional Neural Network (CNN) has become an important tool for machine learning and many related fields [10, 36, 37, 38]. However, due to their nature of computational intensity, as CNNs grow wider and deeper, their memory footprint, power consumption and required floating-point operations (FLOPs) have increased dramatically, thus making them difficult to be deployed on platforms without rich computational resource, like embedded systems. In this context, CNN compression and acceleration methods have been intensively studied, including tensor low rank expansion [31], connection pruning [20], filter pruning [40], quantization [19], knowledge distillation [27], fast convolution [48], feature map compacting [61], etc.

We focus on filter pruning, a.k.a. channel pruning [26] or network slimming [44], for three reasons. Firstly, filter pruning is a universal technique which is able to handle any kinds of CNNs, making no assumptions on the application field, the network architecture or the deployment platform. Secondly, filter pruning effectively reduces the FLOPs of the network, which serve as the main criterion of computational burdens. Lastly, as an important advantage in practice, filter pruning produces a thinner network with no customized structure or extra operation, which is orthogonal to the other model compression and acceleration techniques.

Motivated by the universality and significance, considerable efforts have been devoted to filter pruning techniques. Due to the widely observed redundancy in CNNs [8, 9, 13, 19, 66, 69], numerous excellent works have shown that, if a CNN is pruned appropriately with acceptable structural damage, a follow-up finetuning procedure can restore the performance to a certain degree. 1) Some prior works [2, 5, 28, 40, 49, 50, 66] sort the filters by their importance, directly remove the unimportant ones and re-construct the network with the remaining filters. As the important filters are preserved, a comparable level of performance can be reached by finetuning. However, some recent powerful networks have complicated structures, like identity mapping [23] and dense connection [29], where some layers must be pruned in the same pattern as others, raising an open problem of constrained filter pruning. This further challenges such pruning techniques, as one cannot assume the important filters at different layers reside on the same positions. 2) Obviously, the model is more likely to recover if the destructive impact of pruning is reduced. Taking this into consideration, another family of methods [3, 15, 43, 60, 63] seeks to zero out some filters in advance, where group-

---

*This work is supported by the National Key R&D Program of China (No. 2018YFC0807500), National Natural Science Foundation of China (No. 61571269), National Postdoctoral Program for Innovative Talents (No. BX20180172), and the China Postdoctoral Science Foundation (No. 2018M640131). Corresponding author: Guiguang Ding.

1Here “centripetal” means ”several objects moving towards a center”, not ”an object rotating around a center by the centripetal force”. 
Figure 1: Zeroing-out v.s. centripetal constraint. This figure shows a CNN with 4 and 6 filters at the 1st and 2nd convolutional layer, respectively, which takes a 2-channel input. Left: the 3rd filter at conv1 is zeroed out, thus the 3rd feature map is close to zero, implying that the 3rd input channels of the 6 filters at conv2 are useless. During pruning, the 3rd filters at conv1 along with the 3rd input channels of the 6 filters at conv2 are removed. Right: the 3rd and 4th filters at conv1 are forced to grow close by centripetal constraint until the 3rd and 4th feature maps become identical. But the 3rd and 4th input channels of the 6 filters at conv2 can still grow without constraints, making the encoded information still in full use. When pruned, the 4th filter at conv1 is removed, and the 4th input channel of every filter at conv2 is added to the 3rd channel.

Lasso Regularization [53] is frequently used. Essentially, zeroing filters out can be regarded as producing a desired redundancy pattern in CNNs. After reducing the magnitude of parameters of some whole filters, pruning these filters causes less accuracy drop, hence it becomes easier to restore the performance by finetuning.

In this paper, we also aim to produce some redundancy patterns in CNNs for filter pruning. However, instead of zeroing out filters, which ends up with a pattern where some whole filters are close to zero, we intend to merge multiple filters into one, leading to a redundancy pattern where some filters are identical. The intuition motivating the proposed method is an observation of information flow in CNNs (Fig. 1). 1) If two or more filters are trained to become identical, due to the linearity of convolution, we can simply discard all but leave one filter, and add up the parameters along the corresponding input channels of the next layer. Doing so will cause ZERO performance loss, and there is no need for a time-consuming finetuning process. It is noted that such a finetuning process is essential for the zeroing-out methods [3, 43, 63], as the discarded filters are merely small in magnitude, but still encode a certain quantity of information. Therefore, removing such filters unavoidably degrades the performance of the network. 2) When multiple filters are constrained to grow closer in the parameter hyperspace, which we refer to as the centripetal constraint, though they start to produce increasingly similar information, the information conveyed from the corresponding input channels of the next layer is still in full use, thus the model’s representational capacity is stronger than a counterpart with the filters being zeroed out.

We summarize our contributions as follows.

- We propose Centripetal SGD (C-SGD), an innovative SGD optimization method. As the name suggests, we make multiple filters move towards a center in the hyperspace of the filter parameters. In the meantime, supervised by the model’s original objective function, the performance is maintained as much as possible.
- By C-SGD, we have partly solved constrained filter pruning, an open problem of slimming modern very deep CNNs with complicated structure, where some layers must be pruned in the same pattern as others.
- We have presented both theoretical and empirical analysis of the effectiveness of C-SGD. We have shown empirical evidences supporting our motivation (Fig. 1) and the assumption that the redundancy helps the convergence of neural networks [14, 27]. The codes are available at https://github.com/ShawnDing1994/Centripetal-SGD.

2. Related Work

Filter Pruning. Numerous inspiring works [7, 17, 20, 22, 39, 58, 67] have shown that it is feasible to remove a large portion of connections or neurons from a neural network without a significant performance drop. However, as the connection pruning methods make the parameter tensors no smaller but just sparser, little or no acceleration can be observed without the support from specialized hardware. Then it is natural for researchers to go further on CNNs: by removing filters instead of sporadic connections, we transform the wide convolutional layers into narrower ones, hence the FLOPs, memory footprint and power consumption are significantly reduced. One kind of methods defines the importance of filters by some means, then selects and prunes the unimportant filters carefully to minimize the performance loss. Some prior works measure a filter’s importance by the accuracy reduction (CAR) [2], the channel
contribution variance [50], the Taylor-expansion-based criterion [49], the magnitude of convolution kernels [40] and the average percentage of zero activations (APoZ) [28], respectively; Luo et al. [47] select filters based on the information derived from the next layer; Yu et al. [66] take into consideration the effect of error propagation; He et al. [26] select filters by solving the Lasso regression; He and Han [24] pick up filters with aid of reinforcement learning. Another category seeks to train the network under certain constraints in order to zero out some filters, where group-Lasso regularization is frequently used [3, 43, 63]. It is noteworthy that since removing some whole filters can degrade the network a lot, the CNNs are usually pruned in a layer-by-layer [3, 24, 26, 28, 47, 50] or filter-by-filter [2, 49] manner, and require one or more finetuning processes to restore the accuracy [2, 3, 5, 24, 26, 44, 47, 49, 50, 63, 66].

Other Methods. Apart from filter pruning, some excellent works seek to compress and accelerate CNNs in other ways. Considerable works [4, 14, 31, 32, 54, 56, 65, 68] decompose or approximate the parameter tensors; quantization and binarization techniques [11, 18, 19, 51, 64] approximate a model using fewer bits per parameter; knowledge distillation methods [6, 27, 52] transfer knowledge from a big network to a smaller one; some researchers seek to speed up convolution with the help of perforation [16], FFT [48, 59] or DCT [62]; Wang et al. [61] compact feature maps by extracting information via Circulant matrices. Of note is that since filter pruning simply shrinks a wide CNN into a narrower one with no special structures or extra operations, it is orthogonal to the other methods.

3. Slimming CNNs via Centripetal SGD

3.1. Formulation

In modern CNNs, batch normalization [30] and scaling transformation are commonly used to enhance the representational capacity of convolutional layers. For simplicity and generality, we regard the possible subsequent batch normalization and scaling layer as part of the convolutional layer. Let \( i \) be the layer index, \( M^{(i)} \in \mathbb{R}^{h_i \times w_i \times c_i} \) be an \( h_i \times w_i \) feature map with \( c_i \) channels and \( M^{(i,j)} = M^{(i)}_{:, :, j} \) be the \( j \)-th channel. The convolutional layer \( i \) with kernel size \( u_i \times v_i \) has one \( 4 \)-order tensor and four vectors as parameters at most, namely, \( K^{(i)} \in \mathbb{R}^{u_i \times v_i \times c_{i-1} \times c_i} \) and \( \mu^{(i)}, \sigma^{(i)}, \gamma^{(i)}, \beta^{(i)} \in \mathbb{R}^{c_i} \), where \( K^{(i)} \) is the convolution kernel, \( \mu^{(i)} \) and \( \sigma^{(i)} \) are the mean and standard deviation of batch normalization, \( \gamma^{(i)} \) and \( \beta^{(i)} \) are the parameters of the scaling transformation. Then we use \( P^{(i)} = (K^{(i)}, \mu^{(i)}, \sigma^{(i)}, \gamma^{(i)}, \beta^{(i)}) \) to denote the parameters of layer \( i \). In this paper, the filter \( j \) at layer \( i \) refers to the five-tuple comprising all the parameter slices related to the \( j \)-th output channel of layer \( i \), formally, \( F^{(j)} = (K_{i,,:,,:,j}^{(i)}, \mu_j^{(i)}, \sigma_j^{(i)}, \gamma_j^{(i)}, \beta_j^{(i)}) \). During forward propagation, this layer takes \( M^{(i-1)} \in \mathbb{R}^{h_{i-1} \times w_{i-1} \times c_{i-1}} \) as input and outputs \( M^{(i)} \). Let \( * \) be the 2-D convolution operator, the \( j \)-th output channel is given by

\[
M^{(i,j)} = \sum_{k=1}^{c_{i-1}} M^{(i-1,k)} * K_{i,,:,k,j}^{(i)} - \mu_j^{(i)} + \beta_j^{(i)}.
\]

The importance-based filter pruning methods [2, 28, 40, 49, 50, 66] define the importance of filters by some means, prune the unimportant part and reconstruct the network using the remaining parameters. Let \( I_i \) be the filter index set of layer \( i \) (e.g., \( I_2 = \{1, 2, 3, 4\} \) if the second layer has four filters), \( T \) be the filter importance evaluation function and \( \theta_i \) be the threshold. The remaining set, i.e., the index set of the filters which survive the pruning, is \( R_i = \{ j \in I_i \mid T(F^{(j)}) > \theta_i \} \). Then we reconstruct the network by assembling the parameters sliced from the original tensor or vectors of layer \( i \) into the new parameters. That is,

\[
\hat{P}^{(i)} = (K_{i,,:,:,j}^{(i)}, \mu_j^{(i)}, \sigma_j^{(i)}, \gamma_j^{(i)}, \beta_j^{(i)}).
\]

The input channels of the next layer corresponding to the pruned filters should also be discarded,

\[
\hat{P}^{(i+1)} = (K_{i,,:,:,j}^{(i+1)}, \mu_j^{(i+1)}, \sigma_j^{(i+1)}, \gamma_j^{(i+1)}, \beta_j^{(i+1)}).
\]

3.2. Update Rule

For each convolutional layer, we first divide the filters into clusters. The number of clusters equals the desired number of filters, as we preserve only one filter for each cluster. We use \( C_i \) and \( H \) to denote the set of all filter clusters of layer \( i \) and a single cluster in the form of a filter index set, respectively. We generate the clusters evenly or by k-means [21], between which our experiments demonstrate only minor difference (Table 1).

- **K-means clustering.** We aim to generate clusters with low intra-cluster distance in the parameter hyperspace, such that collapsing them into a single point less impacts the model, which is natural. To this end, we simply flatten the filter’s kernel and use it as the feature vector for k-means clustering.

- **Even clustering.** We can generate clusters with no consideration of the filters’ inherent properties. Let \( c_i \) and \( r_i \) be the number of original filters and desired clusters, respectively, then each cluster will have \([c_i / r_i]\) filters at most. For example, if the second layer has six filters and we wish to slim it to four filters, we will have \( C_2 = \{H_1, H_2, H_3, H_4\} \), where \( H_1 = \{1, 2\}, H_2 = \{3, 4\}, H_3 = \{5\}, H_4 = \{6\} \).

We use \( H(j) \) to denote the cluster containing filter \( j \), so in the above example we have \( H(3) = H_2 \) and \( H(6) = H_4 \). Let \( F^{(j)} \) be the kernel or a vector parameter of filter \( j \), at
each training iteration, the update rule of C-SGD is

\[
F^{(j)} - F^{(j)} + \tau \Delta F^{(j)},
\]

\[
\Delta F^{(j)} = -\frac{\sum_{k \in H(j)} \frac{\partial L}{\partial F^{(j)}}}{|H(j)|} - \eta F^{(j)} + \epsilon \left( \frac{\sum_{k \in H(j)} F^{(k)}}{|H(j)|} - F^{(j)} \right),
\]

(4)

where \( L \) is the original objective function, \( \tau \) is the learning rate, \( \eta \) is the model's original weight decay factor, and \( \epsilon \) is the only introduced hyper-parameter, which is called the centripetal strength.

Let \( L \) be the layer index set, we use the sum of squared kernel deviation \( \chi \) to measure the intra-cluster similarity, i.e., how close filters are in each cluster,

\[
\chi = \sum_{i \in \mathcal{L}} \sum_{j \in \mathcal{I}_i} \| K^{(i)}_{\cdot \cdot ; j} - \frac{\sum_{k \in H(j)} K^{(i)}_{\cdot \cdot ; k}}{|H(j)|} \|^2.
\]

(5)

It is easy to derive from Eq. 4 that if the floating-point operation errors are ignored, \( \chi \) is lowered monotonically and exponentially with a proper learning rate \( \tau \).

The intuition behind Eq. 4 is quite simple: for the filters in the same cluster, the increments derived by the objective function are averaged (the first term), the normal weight decay is the only introduced hyper-parameter, which is called the centripetal strength.

It is easy to derive from Eq. 4 that if the floating-point operation errors are ignored, \( \chi \) is lowered monotonically and exponentially with a proper learning rate \( \tau \).

The intuition behind Eq. 4 is quite simple: for the filters in the same cluster, the increments derived by the objective function are averaged (the first term), the normal weight decay is the only introduced hyper-parameter, which is called the centripetal strength.

In practice, we fix \( \eta \) and reduce \( \tau \) with time just as we do in normal SGD training, and set \( \epsilon \) casually. Intuitively, C-SGD training with a large \( \epsilon \) prefers "rapid change" to "stable transition", and vice versa. If \( \epsilon \) is too large, e.g., 10, the filters are merged in an instant such that the whole process becomes equivalent to training a destroyed model from scratch. If \( \epsilon \) is extremely small, like \( 1 \times 10^{-10} \), the difference between C-SGD training and normal SGD is almost invisible during a long time. However, since the difference among filters in each cluster is reduced monotonically and exponentially, even an extremely small \( \epsilon \) can make the filters close enough, sooner or later. As shown in the Appendix, C-SGD is insensitive to \( \epsilon \).

A simple analogy to weight decay (i.e., \( L2 \) regularization) may help understand Centripetal SGD. Fig. 2a shows a 3-D loss surface, where a certain point \( A \) corresponds to a 2-D parameter \( a = (a_1, a_2) \). Suppose the steepest descent direction is \( AQ_0 \), we have \( AQ_0 = -\frac{\partial L}{\partial a} \), where \( L \) is the objective function. Weight decay is commonly applied to reduce overfitting [35], that is, \( AQ_1 = -\eta a \), where \( \eta \) is the model's weight decay factor, e.g., \( 1 \times 10^{-4} \) for ResNets [23]. The actual gradient descent direction then becomes \( \Delta a = AQ_2 = AQ_0 + AQ_1 = -\frac{\partial L}{\partial a} - \eta a \).

Formally, with \( t \) denoting the number of training iterations, we seek to make point \( A \) and \( B \) grow increasingly 

(a) Normal weight decay. (b) Centripetal constraint.

Figure 2: Gradient descent direction on the loss surface of normal weight decay and centripetal constraint without merging the original gradients.

close and eventually the same by satisfying

\[
\lim_{t \to \infty} \| a^{(t)} - b^{(t)} \| = 0. \tag{6}
\]

Given the fact that \( a^{(t+1)} = a^{(t)} + \tau \Delta a^{(t)} \) and \( b^{(t+1)} = b^{(t)} + \tau \Delta b^{(t)} \), where \( \tau \) is the learning rate, Eq. 6 implies

\[
\lim_{t \to \infty} \| (a^{(t)} - b^{(t)}) + \tau (\Delta a^{(t)} - \Delta b^{(t)}) \| = 0. \tag{7}
\]

We seek to achieve this with \( \lim_{t \to \infty} (\Delta a^{(t)} - \Delta b^{(t)}) = 0 \) as well as \( \lim_{t \to \infty} (a^{(t)} - b^{(t)}) = 0 \). Namely, as two points are growing closer, their gradients should become closer accordingly in order for the training to converge.

If we just wish to make \( A \) and \( B \) closer to each other than they used to be, a natural idea is to push both \( A \) and \( B \) to their midpoint \( \frac{a + b}{2} \), as shown in Fig. 2b. Therefore, the gradient descent direction of point \( A \) becomes

\[
\Delta a = AQ_2 + AQ_3 = -\frac{\partial L}{\partial a} - \eta a + (a + b) - \frac{a + b}{2} \tag{8}
\]

where \( \epsilon \) is a hyper-parameter controlling the intensity or speed of pushing \( A \) and \( B \) close. We have

\[
\Delta b = -\frac{\partial L}{\partial b} - \eta b + \epsilon \left( a + b - \frac{a + b}{2} \right), \tag{9}
\]

\[
\Delta a - \Delta b = \left( \frac{\partial L}{\partial a} - \frac{\partial L}{\partial b} \right) + (\eta + \epsilon)(b - a). \tag{10}
\]

Here we see the problem: we cannot ensure \( \lim_{t \to \infty} \left( \frac{\partial L}{\partial a^{(t)}} - \frac{\partial L}{\partial b^{(t)}} \right) = 0 \). Actually, even \( a = b \) does not imply \( \frac{\partial L}{\partial a} = \frac{\partial L}{\partial b} \), because they participate in different computation flows. As a consequence, we cannot ensure \( \lim_{t \to \infty} (\Delta a^{(t)} - \Delta b^{(t)}) = 0 \) with Eq. 8 and Eq. 9.

We solve this problem by merging the gradients derived from the original objective function. For simplicity and symmetry, by replacing both \( \frac{\partial L}{\partial a} \) in Eq. 8 and \( \frac{\partial L}{\partial b} \) in Eq.
can be easily verified. Intuitively, we deviate both objective-function-related gradients is preserved to main-
the supervision information encoded in the steepest descent direction according to some information of 
and deviate b vice versa, just like the $\ell$-2 regularization deviates both $a$ and $b$ towards the origin of coordinates.

### 3.3. Efficient Implementation of C-SGD

The efficiency of modern CNN training and deployment platforms, e.g., Tensorflow [1], is based on large-scale tensor operations. We therefore seek to implement C-SGD by efficient matrix multiplication which introduces minimal computational burdens. Concretely, given a convolutional layer $i$, the kernel $K \in \mathbb{R}^{h_i \times c_i \times c_{i-1} \times c}$ and the gradient $\frac{\partial L}{\partial K}$, we reshape $K$ to $W \in \mathbb{R}^{h_i \times c_i \times c_{i-1} \times c}$ and $\frac{\partial L}{\partial W}$ accordingly. We construct the averaging matrix $\Gamma = \text{diag}(1)$, $\Lambda = \text{diag}(\eta)$. The other trainable parameters (i.e., $\gamma$ and $\beta$) are reshaped into $W \in \mathbb{R}^{h_i \times c_i}$ and handled in the same way. In practice, we observe almost no difference in the speed between normal SGD and C-SGD using Tensorflow on Nvidia GeForce GTX 1080Ti GPUs with CUDA9.0 and cuDNN7.0.

\[
W \leftarrow W - \tau \left( \frac{\partial L}{\partial W} \Gamma + W \Lambda \right). \tag{11}
\]

\[
\Gamma_{m,n} = \begin{cases} 
\frac{1}{|H(m)|} & \text{if } H(m) = H(n), \\
0 & \text{elsewise} .
\end{cases} \tag{12}
\]

\[
\Lambda_{m,n} = \begin{cases} 
\eta + (1 - 1/|H(m)|) \epsilon & \text{if } H(m) = H(n), \\
0 & \text{elsewise} .
\end{cases} \tag{13}
\]

### 3.4. Filter Trimming after C-SGD

After C-SGD training, since the filters in each cluster have become identical, as will be shown in Sect. 4.3, picking up which one makes no difference. We simply pick up the first filter (i.e., the filter with the smallest index) in each cluster to form the remaining set for each layer, which is

\[ R_i = \{ \text{min}(H) \mid \forall H \in C_i \}. \]

For the next layer, we add the to-be-deleted input channels to the corresponding remaining one,

\[ K^{(i+1)}_{:,k} \leftarrow \sum K^{(i+1)}_{:,H(k):} \quad \forall k \in R_i , \]

then we delete the redundant filters as well as the input channels of the next layer following Eq. 2, 3. Due to the linearity of convolution (Eq. 1), no damage is caused, hence no finetuning is needed.

### 3.5. C-SGD for Constrained Filter Pruning

Recently, accompanied by the advancement of CNN design philosophy, several efficient and compact CNN architectures [23, 29] have emerged and become favored in the real-world applications. Although some excellent works [28, 32, 49, 66, 69] have shown that the classical plain CNNs, e.g., AlexNet [34] and VGG [55], are highly redundant and can be pruned significantly, the pruned versions are usually still inferior to the more up-to-date and complicated CNNs in terms of both accuracy and efficiency. We consider filter pruning for very deep and complicated CNNs challenging for three reasons. 1) Firstly, these networks are designed in consideration of computational efficiency, which makes them inherently compact and efficient. 2) Secondly, these networks are significantly deeper than the classical ones, thus the layer-by-layer pruning techniques become inefficient, and the errors can increase dramatically when propagated through multiple layers, making the estimation of filter importance less accurate [66]. 3) Lastly and most importantly, some innovative structures are heavily used in these networks, e.g., cross-layer connections [23] and dense connections [29], raising an open problem of constrained filter pruning.

\[ i.e. , \] in each stage of ResNets, every residual block is expected to add the learned residuals to the stem feature maps produced by the first or the projection layer (referred to as pacesetter), thus the last layer of every residual block (referred to as follower) must be pruned in the same pattern as the pacesetter, i.e., the remaining set $R$ of all the followers and the pacesetter must be identical, or the network will be damaged so badly that finetuning cannot restore its accuracy. For example, Li et al. [40] once tried violently pruning ResNets but resulted in low accuracy. In some successful explorations, Li et al. [40] sidestep this problem by only pruning the internal layers on ResNet-56, i.e., the first layers in each residual block. Liu et al. [44] and He et al. [26] skip pruning these troublesome layers and insert an extra sampler layer before the first layer in each residual block during inference time to reduce the input channels. Though these methods are able to prune the networks to some extent, from a holistic perspective the networks are not literally “slimmed” but actually “clipped”, as shown in Fig. 3.

We have partly solved this open problem by C-SGD, where the key is to force different layers to learn the same redundancy pattern. For example, if the layer $p$ and $q$ have to be pruned in the same pattern, we only generate clusters for the layer $p$ by some means and assign the resulting cluster set to the layer $q$, namely, $C_q \leftarrow C_p$. Then during C-SGD training, the same redundancy patterns among filters in both layer $p$ and $q$ are produced. \textit{i.e.}, if the $j$-th and $k$-th filters...
at layer $p$ become identical, we ensure the sameness of the $j$-th and $k$-th filters at layer $q$ as well, thus the troublesome layers can be pruned along with the others. Some sketches are presented in the Appendix for more intuitions.

4. Experiments

4.1. Slimming Very Deep and Complicated CNNs

We experiment on CIFAR-10 [33] and ImageNet-1K [12] to evaluate our method. For each trial we start from a well-trained base model and apply C-SGD training on all the target layers simultaneously. The comparisons between C-SGD and other filter pruning methods are presented in Table 1 and Table 2 in terms of both absolute and relative error increase, which are commonly adopted as the metrics to fairly compare the change of accuracy on different base models. E.g., the Top-1 accuracy of our ResNet-50 base model and C-SGD-70 is 75.33% and 75.27%, thus the absolute and relative error increase is $75.33\% - 75.27\% = 0.06\%$ and $\frac{0.06}{0.75} = 0.24\%$, respectively.

CIFAR-10. The base models are trained from scratch for 600 epochs to ensure the convergence, which is much longer than the usually adopted benchmarks (160 [23] or 300 [29] epochs), such that the improved accuracy of the pruned model cannot be simply attributed to the extra training epochs on a base model which has not fully converged. We use the data augmentation techniques adopted by [23], i.e., padding to $40 \times 40$, random cropping and flipping. The hyper-parameter $\epsilon$ is casually set to $3 \times 10^{-3}$. We perform C-SGD training with batch size 64 and a learning rate initialized to $3 \times 10^{-2}$ then decayed by 0.1 when the loss stops decreasing. For each network we perform two experiments independently, where the only difference is the way we generate filter clusters, namely, even dividing or k-means clustering. We seek to reduce the FLOPs of every model by around 60%, so we prune $3/8$ of every convolutional layer of ResNets, thus the parameters and FLOPs are reduced by around $1 - (5/8)^3 = 61\%$. Aggressive as it is, no obvious accuracy drop is observed. For DenseNet-40, the pruned model has 5, 8 and 10 incremental convolutional layers in the three stages, respectively, so that the FLOPs is reduced by 60.05%, and a significantly increased accuracy is observed, which is consistent with but better than that of [44].

ImageNet. We perform experiments using ResNet-50 [23] on ImageNet to validate the effectiveness of C-SGD on the real-world applications. We apply k-means clustering on the filter kernels to generate the clusters, then use the ILSVRC2015 training set which contains 1.28M high-quality images for training. We adopt the standard data augmentation techniques including b-box distortion and color shift. At test time, we use a single central crop. For C-SGD-7/10, C-SGD-6/10 and C-SGD-5/10, all the first and second layers in each residual block are shrunk to 70%, 60% and 50% of the original width, respectively.

Discussions. Our pruned networks exhibit fewer FLOPs, simpler structures and higher or comparable accuracy. Note that we apply the same pruning ratio globally for ResNets, and better results are promising to be achieved if more layer sensitivity analyzing experiments[26, 40, 66] are conducted, and the resulting network structures are tuned accordingly. Interestingly, even arbitrarily generated clusters can produce reasonable results (Table 1).

4.2. Redundant Training vs. Normal Training

The comparisons between C-SGD and other pruning-and-finetuning methods [26, 40, 47, 66] indicate that it may be better to train a redundant network and equivalently transform it to a narrower one than to finetune it after pruning. This observation is consistent with [14] and [27], where the authors believe that the redundancy in neural networks is necessary to overcome a highly non-convex optimization.

We verify this assumption by training a narrow CNN with normal SGD and comparing it with another model trained using C-SGD with the equivalent width, which means that some redundant filters are produced during training and trimmed afterwards, resulting in the same network structure as the normally trained model. For example, if a network has $2 \times$ number of filters as the normal counterpart but every two filters are identical, they will end up with the same structure. If the redundant one outperforms the normal one, we can conclude that C-SGD does yield more powerful networks by exploiting the redundant filters.

On DenseNet-40, we evenly divide the 12 filters at each incremental layer into 3 clusters, use C-SGD to train the network from scratch, then trim it to obtain a DenseNet-40 with 3 filters per incremental layer, i.e., during training, every 4 filters are growing centripetally. As contrast, we train a DenseNet-40 with originally 3 filters per layer by normal SGD. Another group of experiments where each layer ends up with 6 filters are carried out similarly. After that, ex-
Table 1: Pruning Results on CIFAR-10. For C-SGD, the left is achieved by even clustering, and the right uses k-means.

<table>
<thead>
<tr>
<th>Model</th>
<th>Result</th>
<th>Base Top1</th>
<th>Pruned Top1</th>
<th>K-means Top1 error</th>
<th>FLOPs ↓%</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-56</td>
<td>Li et al. [40]</td>
<td>93.04</td>
<td>93.06</td>
<td>-0.02 / -0.28</td>
<td>27.60</td>
<td>only internals pruned</td>
</tr>
<tr>
<td>ResNet-56</td>
<td>NISP-56 [66]</td>
<td>-</td>
<td>-</td>
<td>0.03 / -</td>
<td>43.61</td>
<td>-</td>
</tr>
<tr>
<td>ResNet-56</td>
<td>Channel Pruning [26]</td>
<td>92.8</td>
<td>91.8</td>
<td>1.0 / 13.88</td>
<td>50</td>
<td>sampler layer</td>
</tr>
<tr>
<td>ResNet-56</td>
<td>ADC [24]</td>
<td>92.8</td>
<td>91.9</td>
<td>0.9 / 12.5</td>
<td>50</td>
<td>sampler layer</td>
</tr>
<tr>
<td><strong>ResNet-56</strong></td>
<td>C-SGD-5/8</td>
<td><strong>93.39</strong></td>
<td><strong>93.31 / 93.44</strong></td>
<td><strong>-0.05 / -0.75</strong></td>
<td><strong>60.85</strong></td>
<td><strong>10-20-40</strong></td>
</tr>
<tr>
<td>ResNet-110</td>
<td>Li et al. [40]</td>
<td>93.53</td>
<td>93.30</td>
<td>0.23 / 3.55</td>
<td>38.60</td>
<td>only internals pruned</td>
</tr>
<tr>
<td>ResNet-110</td>
<td>NISP-110 [66]</td>
<td>-</td>
<td>-</td>
<td>0.18 / -</td>
<td>43.78</td>
<td>-</td>
</tr>
<tr>
<td><strong>ResNet-110</strong></td>
<td>C-SGD-5/8</td>
<td><strong>94.38</strong></td>
<td><strong>94.44 / 94.27</strong></td>
<td><strong>0.11 / 1.95</strong></td>
<td><strong>60.89</strong></td>
<td><strong>10-20-40</strong></td>
</tr>
<tr>
<td>ResNet-164</td>
<td>Network Slimming [44]</td>
<td>94.58</td>
<td>94.73</td>
<td>-0.15 / -2.76</td>
<td>44.90</td>
<td>sampler layer</td>
</tr>
<tr>
<td><strong>ResNet-164</strong></td>
<td>C-SGD-5/8</td>
<td><strong>94.83</strong></td>
<td><strong>94.75 / 94.75</strong></td>
<td><strong>0.08 / 1.54</strong></td>
<td><strong>60.91</strong></td>
<td><strong>10-20-40</strong></td>
</tr>
<tr>
<td>DenseNet-40</td>
<td>Network Slimming [44]</td>
<td>93.89</td>
<td>94.35</td>
<td>-0.46 / -7.52</td>
<td>55.00</td>
<td>-</td>
</tr>
<tr>
<td><strong>DenseNet-40</strong></td>
<td>C-SGD-5-8-10</td>
<td><strong>93.81</strong></td>
<td><strong>94.31 / 94.44</strong></td>
<td><strong>-0.63 / -10.17</strong></td>
<td><strong>60.05</strong></td>
<td><strong>5-8-10</strong></td>
</tr>
</tbody>
</table>

Table 2: Pruning ResNet-50 on ImageNet using k-means clustering.

<table>
<thead>
<tr>
<th>Result</th>
<th>Base Top1</th>
<th>Base Top5</th>
<th>Pruned Top1</th>
<th>Pruned Top5</th>
<th>Top1 Error Abs/Rel ↑%</th>
<th>Top5 error Abs/Rel ↑%</th>
<th>FLOPs ↓%</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-SGD-70</td>
<td>75.33</td>
<td>92.56</td>
<td>75.27</td>
<td>92.46</td>
<td>0.06 / 0.24</td>
<td>0.10 / 1.34</td>
<td>36.75</td>
</tr>
<tr>
<td>ThiNet-70 [47]</td>
<td>72.88</td>
<td>91.14</td>
<td>72.04</td>
<td>90.67</td>
<td>0.84 / 3.09</td>
<td>0.47 / 5.30</td>
<td>36.75</td>
</tr>
<tr>
<td>SFP [25]</td>
<td>76.15</td>
<td>92.87</td>
<td>74.61</td>
<td>92.06</td>
<td>1.54 / 6.45</td>
<td>0.81 / 9.16</td>
<td>41.8</td>
</tr>
<tr>
<td>NISP [66]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>- / -</td>
<td>- / -</td>
<td>43.82</td>
</tr>
<tr>
<td>C-SGD-60</td>
<td>75.33</td>
<td>92.56</td>
<td>74.93</td>
<td>92.27</td>
<td>0.40 / 1.62</td>
<td>0.29 / 3.89</td>
<td>46.24</td>
</tr>
<tr>
<td>Channel Pruning [26]</td>
<td>-</td>
<td>92.2</td>
<td>73.4</td>
<td>91.4</td>
<td>1.9 / 7.69</td>
<td>0.8 / 10.25</td>
<td>49.6</td>
</tr>
<tr>
<td>Autopruner [46]</td>
<td>76.15</td>
<td>92.87</td>
<td>74.76</td>
<td>92.15</td>
<td>1.39 / 5.82</td>
<td>0.72 / 10.09</td>
<td>51.21</td>
</tr>
<tr>
<td>GDP [42]</td>
<td>75.13</td>
<td>92.30</td>
<td>71.89</td>
<td>90.71</td>
<td>3.24 / 13.02</td>
<td>1.59 / 20.64</td>
<td>51.30</td>
</tr>
<tr>
<td>SSR-L2 [41]</td>
<td>75.12</td>
<td>92.30</td>
<td>71.47</td>
<td>90.19</td>
<td>3.65 / 14.67</td>
<td>2.11 / 27.40</td>
<td>55.76</td>
</tr>
<tr>
<td>DCP [70]</td>
<td>76.01</td>
<td>92.93</td>
<td>74.95</td>
<td>92.32</td>
<td>1.06 / 4.41</td>
<td>0.61 / 8.62</td>
<td>55.76</td>
</tr>
<tr>
<td>ThiNet-50 [47]</td>
<td>72.88</td>
<td>91.14</td>
<td>71.01</td>
<td>90.02</td>
<td>1.87 / 6.89</td>
<td>1.12 / 12.64</td>
<td>55.76</td>
</tr>
<tr>
<td><strong>C-SGD-50</strong></td>
<td><strong>75.33</strong></td>
<td><strong>92.56</strong></td>
<td><strong>74.54</strong></td>
<td><strong>92.09</strong></td>
<td><strong>0.79 / 3.20</strong></td>
<td><strong>0.47 / 6.31</strong></td>
<td><strong>55.76</strong></td>
</tr>
</tbody>
</table>

4.3. Making Filters Identical vs. Zeroing Out

As making filters identical and zeroing filters out [3, 15, 43, 60, 63] are two means of producing redundancy patterns for filter pruning, we perform controlled experiments on VGG [55] are also carried out, where we slim each layer to 1/4 and 1/2 of the original width, respectively. It can be concluded from Table 3 that the redundant filters do help, compared to a normally trained counterpart with the equivalent width. This observation supports our intuition that the centripetally growing filters can maintain the model’s representational capacity to some extent because though these filters are constrained, their corresponding input channels are still in full use and can grow without constraints (Fig. 1).

Table 3: Validation accuracy of scratch-trained DenseNet-40 and VGG using C-SGD or normal SGD on CIFAR-10.

<table>
<thead>
<tr>
<th>Model</th>
<th>Normal SGD</th>
<th>C-SGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>DenseNet-3</td>
<td>88.60</td>
<td>89.96</td>
</tr>
<tr>
<td>DenseNet-6</td>
<td>89.96</td>
<td>90.89</td>
</tr>
<tr>
<td>VGG-1/4</td>
<td>90.16</td>
<td>90.64</td>
</tr>
<tr>
<td>VGG-1/2</td>
<td>92.49</td>
<td>93.22</td>
</tr>
</tbody>
</table>
Lasso, 3/8 of the original filters in the pacesetters and internal layers are regularized by group-Lasso, and the followers are handled in the same pattern. We use the aforementioned sum of squared kernel deviation $\chi$ and the sum of squared kernel residuals $\phi$ as follows to measure the redundancy, respectively. Let $\mathcal{L}$ be the layer index set and $\mathcal{P}_i$ be the to-be-pruned filter set of layer $i$, i.e., the set of the 3/8 filters with group-Lasso regularization,

$$\phi = \sum_{i \in \mathcal{L}} \sum_{j \in \mathcal{P}_i} ||K^{(i)}_{\cdot \cdot \cdot j}||_2^2.$$

We present in Fig. 4 the curves of $\chi$, $\phi$ as well as the validation accuracy both before and after pruning. The learning rate $\tau$ is initially set to $3 \times 10^{-2}$ and decayed by 0.1 at epoch 100 and 200, respectively. It can be observed that: 1) Group Lasso cannot literally zero out filters, but can decrease their magnitude to some extent, as $\phi$ plateaus when the gradients derived from the regularization term become close to those derived from the original objective function. We empirically find out that even when $\phi$ reaches around $4 \times 10^{-4}$, which is nearly $2 \times 10^6$ times smaller than the initial value, pruning still causes obvious damage (around 10% accuracy drop). When the learning rate is decayed and $\phi$ is reduced at epoch 200, we observe no improvement in the pruned accuracy, therefore no more experiments with smaller learning rate or stronger group-Lasso regularization are conducted. We reckon this is due to the error propagation and amplification in very deep CNNs [66]. 2) By C-SGD, $\chi$ is reduced monotonically and perfectly exponentially, which leads to faster convergence. I.e., the filters in each cluster can become infinitely close to each other at a constant rate with a constant learning rate. For C-SGD, pruning causes absolutely no performance loss after around 90 epochs. 3) Training with group-Lasso is $2 \times$ slower than C-SGD as it requires costly square root operations.

**4.4. C-SGD vs. Other Filter Pruning Methods**

We compare C-SGD with other methods by controlled experiments on DenseNet-40 [29]. We slim every incremental layer of a well-trained DenseNet-40 to 3 and 6 filters, respectively. The experiments are repeated 3 times, and all the results are presented in Fig. 5. The training setting is kept the same for every model: learning rate $\tau = 3 \times 10^{-3}, 3 \times 10^{-4}, 3 \times 10^{-5}, 3 \times 10^{-6}$ for 200, 200, 100 and 100 epochs, respectively, to ensure the convergence of every model. For our method, the models are trained with C-SGD and trimmed. For Magnitude- [40], APoZ- [28] and Taylor-expansion-based [49], the models are pruned by different criteria and finetuned. The models labeled as Lasso are trained with group-Lasso Regularization for 600 epochs in advance, pruned, then finetuned for another 600 epochs with the same learning rate schedule, so that the comparison is actually biased towards the Lasso method. The models are tested on the validation set every 10,000 iterations (12.8 epochs). The results reveal the superiority of C-SGD in terms of higher accuracy and also the better stability. Though group-Lasso Regularization can indeed reduce the performance drop caused by pruning, it is outperformed by C-SGD by a large margin. It is interesting that the violently pruned networks are unstable and easily trapped in the local minimum, e.g., the accuracy curves increase steeply in the beginning but slightly decline afterwards. This observation is consistent with that of Liu et al. [45].

**5. Conclusion**

We have proposed to produce identical filters in CNNs for network slimming. The intuition is that making filters identical can not only eliminate the need for finetuning but also preserve more representational capacity of the network, compared to the zeroing-out fashion (Fig. 1). We have partly solved an open problem of constrained filter pruning on very deep and complicated CNNs and achieved state-of-the-art results on several common benchmarks. By training networks with redundant filters using C-SGD, we have demonstrated empirical evidences for the assumption that redundancy can help the convergence of neural network training, which may encourage future studies. Apart from pruning, we consider C-SGD promising to be applied as a means of regularization or training technique.
References


