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.

[1] Here "centripetal" means "several objects moving towards a center", not "an object rotating around a center by the centripetal force".

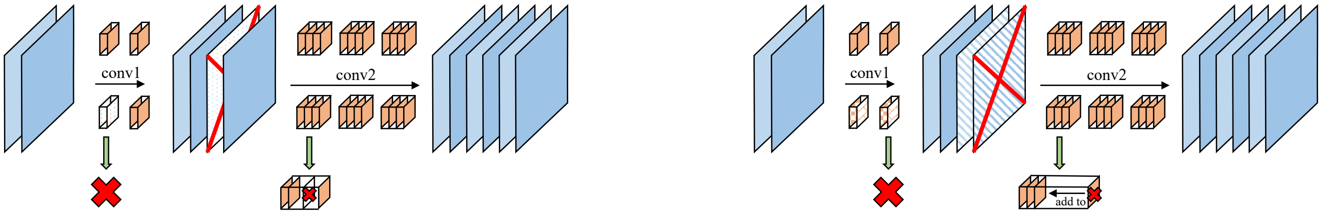


Figure 1: Zeroing-out v.s. centripetal constraint. This figure shows a CNN with 4 and 6 filters at the 1st and 2nd convolutional layer, respectively, which takes a 2-channel input. Left: the 3rd filter at conv1 is zeroed out, thus the 3rd feature map is close to zero, implying that the 3rd input channels 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 to produce redundancy patterns in CNNs by training some filters to become identical. Compared to the importance-based filter pruning methods, doing so requires no heuristic knowledge about the importance of filter. Compared to the zeroing-out methods, no finetuning is needed, and more representational capacity of the network is preserved.
- 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, 28, 40, 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, $\boldsymbol{M}^{(i)} \in \mathbb{R}^{h_i \times w_i \times c_i}$ be an $h_i \times w_i$ feature map with c_i channels and $\boldsymbol{M}^{(i,j)} = \boldsymbol{M}^{(i)}_{:,:,j}$ be the j-th channel. The convolutional layer i with kernel size $u_i \times v_i$ has one 4th-order tensor and four vectors as parameters at most, namely, $\boldsymbol{K}^{(i)} \in \mathbb{R}^{u_i \times v_i \times c_{i-1} \times c_i}$ and $\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)}, \boldsymbol{\gamma}^{(i)}, \boldsymbol{\beta}^{(i)} \in \mathbb{R}^{c_i}$, where $\boldsymbol{K}^{(i)}$ is the convolution kernel, $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$ are the mean and standard deviation of batch normalization, $\boldsymbol{\gamma}^{(i)}$ and $\boldsymbol{\beta}^{(i)}$ are the parameters of the scaling transformation. Then we use $\boldsymbol{P}^{(i)} = (\boldsymbol{K}^{(i)}, \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)}, \boldsymbol{\gamma}^{(i)}, \boldsymbol{\beta}^{(i)})$ 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, $\boldsymbol{F}^{(j)} = (\boldsymbol{K}^{(i)}_{:,:,:,j}, \mu^{(i)}_j, \sigma^{(i)}_j, \gamma^{(i)}_j, \beta^{(i)}_j)$. During forward propagation, this layer takes $\boldsymbol{M}^{(i-1)} \in \mathbb{R}^{h_{i-1} \times w_{i-1} \times c_{i-1}}$ as input and outputs $\boldsymbol{M}^{(i)}$. Let $*$ be the 2-D convolution operator, the j-th output channel is given by

$$\boldsymbol{M}^{(i,j)} = \frac{\sum_{k=1}^{c_{i-1}} \boldsymbol{M}^{(i-1,k)} * \boldsymbol{K}^{(i)}_{:,:,k,j} - \mu^{(i)}_j}{\sigma^{(i)}_j} \gamma^{(i)}_j + \beta^{(i)}_j \,. \tag{1}$$

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 $\mathcal{I}_i$ be the filter index set of layer i (*e.g.*, $\mathcal{I}_2 = \{1, 2, 3, 4\}$ if the second layer has four filters), T be the filter importance evaluation function and θ_i be the threshold. The remaining set, *i.e.*, the index set of the filters which survive the pruning, is $\mathcal{R}_i = \{j \in \mathcal{I}_i \mid T(\boldsymbol{F}^{(j)}) > \theta_i\}$. 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,

$$\hat{\boldsymbol{P}}^{(i)} = (\boldsymbol{K}^{(i)}_{:,:,:,\mathcal{R}_i}, \boldsymbol{\mu}^{(i)}_{\mathcal{R}_i}, \boldsymbol{\sigma}^{(i)}_{\mathcal{R}_i}, \boldsymbol{\gamma}^{(i)}_{\mathcal{R}_i}, \boldsymbol{\beta}^{(i)}_{\mathcal{R}_i}) \,. \tag{2}$$

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

$$\hat{\boldsymbol{P}}^{(i+1)} = (\boldsymbol{K}^{(i+1)}_{:,:,\mathcal{R}_i,:}, \boldsymbol{\mu}^{(i+1)}, \boldsymbol{\sigma}^{(i+1)}, \boldsymbol{\gamma}^{(i+1)}, \boldsymbol{\beta}^{(i+1)}) \,. \tag{3}$$

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 $\mathcal{C}_i$ and $\mathcal{H}$ 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 $\lceil c_i/r_i \rceil$ filters at most. For example, if the second layer has six filters and we wish to slim it to four filters, we will have $\mathcal{C}_2 = \{\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4\}$, where $\mathcal{H}_1 = \{1, 2\}, \mathcal{H}_2 = \{3, 4\}, \mathcal{H}_3 = \{5\}, \mathcal{H}_4 = \{6\}$.

We use $H(j)$ to denote the cluster containing filter j, so in the above example we have $H(3) = \mathcal{H}_2$ and $H(6) = \mathcal{H}_4$. Let $\boldsymbol{F}^{(j)}$ be the kernel or a vector parameter of filter j, at each training iteration, the update rule of C-SGD is

$$\begin{aligned} \boldsymbol{F}^{(j)} &\leftarrow \boldsymbol{F}^{(j)} + \tau\Delta\boldsymbol{F}^{(j)}\,, \\ \Delta\boldsymbol{F}^{(j)} &= -\frac{\sum_{k\in H(j)}\frac{\partial L}{\partial \boldsymbol{F}^{(k)}}}{|H(j)|} - \eta\boldsymbol{F}^{(j)} \\ &\quad + \epsilon\Big(\frac{\sum_{k\in H(j)}\boldsymbol{F}^{(k)}}{|H(j)|} - \boldsymbol{F}^{(j)}\Big)\,, \end{aligned} \tag{4}$$

where L is the original objective function, τ is the learning rate, η is the model's original weight decay factor, and ϵ is the only introduced hyper-parameter, which is called the *centripetal strength*.

Let $\mathcal{L}$ be the layer index set, we use the *sum of squared kernel deviation* χ to measure the intra-cluster similarity, *i.e.*, how close filters are in each cluster,

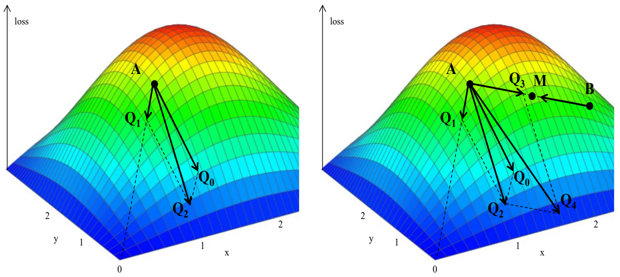
$$\chi = \sum_{i\in\mathcal{L}}\sum_{j\in\mathcal{I}_i} ||\boldsymbol{K}^{(i)}_{:,:,:,j} - \frac{\sum_{k\in H(j)}\boldsymbol{K}^{(i)}_{:,:,:,k}}{|H(j)|}||_2^2\,. \tag{5}$$

It is easy to derive from Eq. 4 that if the floating-point operation errors are ignored, χ is lowered *monotonically* and *exponentially* with a proper learning rate τ.

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 applied as well (the second term), and the difference in the initial values is gradually eliminated (the last term), so the filters will move towards their center in the hyperspace.

In practice, we fix η and reduce τ with time just as we do in normal SGD training, and set ϵ casually. Intuitively, C-SGD training with a large ϵ prefers "rapid change" to "stable transition", and vice versa. If ϵ 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 ϵ is extremely small, like 1×10^{-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 ϵ can make the filters close enough, sooner or later. As shown in the Appendix, C-SGD is insensitive to ϵ.

A simple analogy to weight decay (*i.e.*, ℓ-2 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 $\boldsymbol{a} = (a_1, a_2)$. Suppose the steepest descent direction is $\overrightarrow{AQ_0}$, we have $\overrightarrow{AQ_0} = -\frac{\partial L}{\partial \boldsymbol{a}}$, where L is the objective function. Weight decay is commonly applied to reduce overfitting [35], that is, $\overrightarrow{AQ_1} = -\eta\boldsymbol{a}$, where η is the model's weight decay factor, *e.g.*, 1×10^{-4} for ResNets [23]. The actual gradient descent direction then becomes $\Delta\boldsymbol{a} = \overrightarrow{AQ_2} = \overrightarrow{AQ_0} + \overrightarrow{AQ_1} = -\frac{\partial L}{\partial \boldsymbol{a}} - \eta\boldsymbol{a}$.



(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.

Formally, with t denoting the number of training iterations, we seek to make point A and B grow increasingly close and eventually the same by satisfying

$$\lim_{t\to\infty} ||\boldsymbol{a}^{(t)} - \boldsymbol{b}^{(t)}|| = 0\,. \tag{6}$$

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

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

We seek to achieve this with $\lim\limits_{t\to\infty}(\Delta\boldsymbol{a}^{(t)} - \Delta\boldsymbol{b}^{(t)}) = \boldsymbol{0}$ as well as $\lim\limits_{t\to\infty}(\boldsymbol{a}^{(t)} - \boldsymbol{b}^{(t)}) = \boldsymbol{0}$. 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 $M(\frac{\boldsymbol{a}+\boldsymbol{b}}{2})$, as shown in Fig. 2b. Therefore, the gradient descent direction of point A becomes

$$\Delta\boldsymbol{a} = \overrightarrow{AQ_2} + \overrightarrow{AQ_3} = -\frac{\partial L}{\partial \boldsymbol{a}} - \eta\boldsymbol{a} + \epsilon(\frac{\boldsymbol{a}+\boldsymbol{b}}{2} - \boldsymbol{a})\,, \tag{8}$$

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

$$\Delta\boldsymbol{b} = -\frac{\partial L}{\partial \boldsymbol{b}} - \eta\boldsymbol{b} + \epsilon(\frac{\boldsymbol{a}+\boldsymbol{b}}{2} - \boldsymbol{b})\,, \tag{9}$$

$$\Delta\boldsymbol{a} - \Delta\boldsymbol{b} = (\frac{\partial L}{\partial \boldsymbol{b}} - \frac{\partial L}{\partial \boldsymbol{a}}) + (\eta + \epsilon)(\boldsymbol{b} - \boldsymbol{a})\,. \tag{10}$$

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

9 with $\frac{1}{2}(\frac{\partial L}{\partial \boldsymbol{a}} + \frac{\partial L}{\partial \boldsymbol{b}})$, we have $\Delta \boldsymbol{a} - \Delta \boldsymbol{b} = (\eta + \epsilon)(\boldsymbol{b} - \boldsymbol{a})$. In this way, the supervision information encoded in the objective-function-related gradients is preserved to maintain the model's performance, and Eq. 6 is satisfied, which can be easily verified. Intuitively, we deviate $\boldsymbol{a}$ from the steepest descent direction according to some information of $\boldsymbol{b}$ and deviate $\boldsymbol{b}$ vice versa, just like the ℓ-2 regularization deviates both $\boldsymbol{a}$ and $\boldsymbol{b}$ 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 $\boldsymbol{K} \in \mathbb{R}^{u_i \times v_i \times c_{i-1} \times c_i}$ and the gradient $\frac{\partial L}{\partial \boldsymbol{K}}$, we reshape $\boldsymbol{K}$ to $\boldsymbol{W} \in \mathbb{R}^{u_i v_i c_{i-1} \times c_i}$ and $\frac{\partial L}{\partial \boldsymbol{K}}$ to $\frac{\partial L}{\partial \boldsymbol{W}}$ accordingly. We construct the averaging matrix $\boldsymbol{\Gamma} \in \mathbb{R}^{c_i \times c_i}$ and decaying matrix $\boldsymbol{\Lambda} \in \mathbb{R}^{c_i \times c_i}$ as Eq. 12 and Eq. 13 such that Eq. 11 is equivalent to Eq. 4, which can be easily verified. Obviously, when the number of clusters equals that of the filters, Eq. 11 degrades into normal SGD with $\boldsymbol{\Gamma} = diag(1), \boldsymbol{\Lambda} = diag(\eta)$. The other trainable parameters (*i.e.*, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$) are reshaped into $\boldsymbol{W} \in \mathbb{R}^{1 \times c_i}$ 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.

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \tau(\frac{\partial L}{\partial \boldsymbol{W}}\boldsymbol{\Gamma} + \boldsymbol{W}\boldsymbol{\Lambda})\,. \tag{11}$$

$$\boldsymbol{\Gamma}_{m,n} = \begin{cases} 1/|H(m)| & \text{if } H(m) = H(n)\,, \\ 0 & \text{elsewise}\,. \end{cases} \tag{12}$$

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

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

$$\mathcal{R}_i = \{min(\mathcal{H}) \mid \forall \mathcal{H} \in \mathcal{C}_i\}.$$

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

$$\boldsymbol{K}^{(i+1)}_{:,:,k,:} \leftarrow \sum \boldsymbol{K}^{(i+1)}_{:,:,H(k),:} \quad \forall k \in \mathcal{R}_i\,,$$

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.

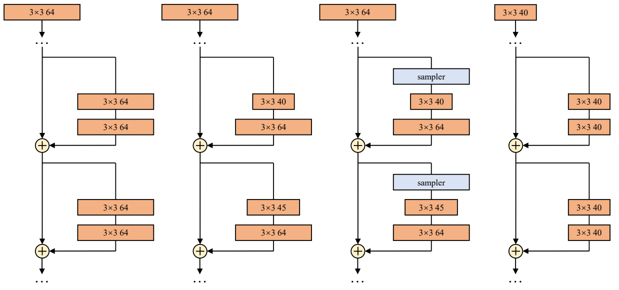
3.5. C-SGD for Constrained Filter Pruning

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. Altough 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 $\mathcal{R}$ 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, $\mathcal{C}_q \leftarrow \mathcal{C}_p$. Then during C-SGD training, the same redundancy patterns among filters in both layer p and q are produced. *I.e.*, if the j-th and k-th filters



(a) Original. (b) Clipped. (c) Sampled. (d) Slimmed.

Figure 3: Compared to prior workings which only clip the internal layers [40] or insert sampler layers [26, 44] on ResNets, C-SGD is literally "slimming" the network.

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 $\frac{0.06}{100-75.33} = 0.24\%$, 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 40×40, random cropping and flipping. The hyper-parameter ϵ is casually set to 3×10^{-3}. We perform C-SGD training with batch size 64 and a learning rate initialized to 3×10^{-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 $1 - (5/8)^2 = 61\%$. 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× 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.

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

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

Result	Base Top1	Base Top5	Pruned Top1	Pruned Top5	Top1 Error Abs/Rel ↑%	Top5 error Abs/Rel ↑%	FLOPs ↓%
C-SGD-70	**75.33**	**92.56**	**75.27**	**92.46**	**0.06 / 0.24**	**0.10 / 1.34**	**36.75**
ThiNet-70 [47]	72.88	91.14	72.04	90.67	0.84 / 3.09	0.47 / 5.30	36.75
SFP [25]	76.15	92.87	74.61	92.06	1.54 / 6.45	0.81 / 11.36	41.8
NISP [66]	-	-	-	-	0.89 / -	- / -	43.82
C-SGD-60	**75.33**	**92.56**	**74.93**	**92.27**	**0.40 / 1.62**	**0.29 / 3.89**	**46.24**
CFP [57]	75.3	92.2	73.4	91.4	1.9 / 7.69	0.8 / 10.25	49.6
Channel Pruning [26]	-	92.2	-	90.8	- / -	1.4 / 17.94	50
Autopruner [46]	76.15	92.87	74.76	92.15	1.39 / 5.82	0.72 / 10.09	51.21
GDP [42]	75.13	92.30	71.89	90.71	3.24 / 13.02	1.59 / 20.64	51.30
SSR-L2 [41]	75.12	92.30	71.47	90.19	3.65 / 14.67	2.11 / 27.40	55.76
DCP [70]	76.01	92.93	74.95	92.32	1.06 / 4.41	0.61 / 8.62	55.76
ThiNet-50 [47]	72.88	91.14	71.01	90.02	1.87 / 6.89	1.12 / 12.64	55.76
C-SGD-50	**75.33**	**92.56**	**74.54**	**92.09**	**0.79 / 3.20**	**0.47 / 6.31**	**55.76**

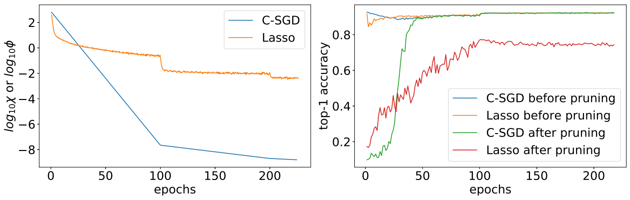
periments on VGG [55] are also carried out, where we slim each layer to 1/4 and 1/2 of the original width, respectively. It can be concluded from Table. 3 that the redundant filters do help, compared to a normally trained counterpart with the equivalent width. This observation supports our intuition 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.

Model	Normal SGD	C-SGD
DenseNet-3	88.60	**89.96**
DenseNet-6	89.96	**90.89**
VGG-1/4	90.16	**90.64**
VGG-1/2	92.49	**93.22**

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 ResNet-56 to investigate the difference. For fair comparison, we aim to produce the same number of redundant filters in both the model trained with C-SGD and the one with group-Lasso Regularization [53]. For C-SGD, the number of clusters in each layer is 5/8 of the number of filters. For



(a) Values of χ or ϕ.

(b) Validation accuracy.

Figure 4: Training process with C-SGD or group-Lasso on ResNet-56. Note the logarithmic scale of the left figure.

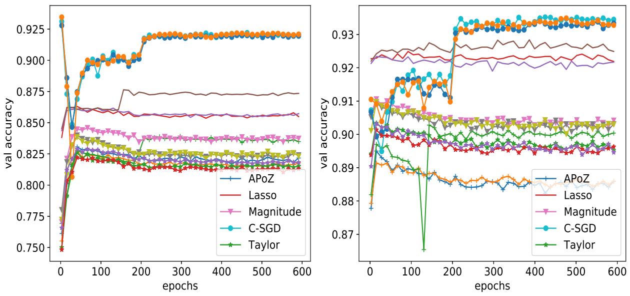
Lasso, 3/8 of the original filters in the pacesetters and internal layers are regularized by group-Lasso, and the followers are handled in the same pattern. We use the aforementioned sum of squared kernel deviation χ and the *sum of squared kernel residuals* ϕ as follows to measure the redundancy, respectively. 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,

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

We present in Fig. 4 the curves of χ, ϕ as well as the validation accuracy both before and after pruning. The learning rate τ is initially set to 3×10^{-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 ϕ 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 ϕ reaches around 4×10^{-4}, which is nearly 2×10^6 times smaller than the initial value, pruning still causes obvious damage (around 10% accuracy drop). When the learning rate is decayed and ϕ 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, χ 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× 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].



(a) Three filters per layer.

(b) Six filters per layer.

Figure 5: Controlled pruning experiments on DenseNet-40.

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

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] R. Abbasi-Asl and B. Yu. Structural compression of convolutional neural networks based on greedy filter pruning. *arXiv preprint arXiv:1705.07356*, 2017.

[3] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[4] J. M. Alvarez and M. Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867, 2017.

[5] S. Anwar, K. Hwang, and W. Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017.

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

[7] G. Castellano, A. M. Fanelli, and M. Pelillo. An iterative pruning algorithm for feedforward neural networks. *IEEE transactions on Neural networks*, 8(3):519–531, 1997.

[8] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2857–2865, 2015.

[9] M. D. Collins and P. Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

[10] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[11] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[13] M. Denil, B. Shakibi, L. Dinh, N. De Freitas, et al. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pages 2148–2156, 2013.

[14] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[15] X. Ding, G. Ding, J. Han, and S. Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[16] M. Figurnov, A. Ibraimova, D. P. Vetrov, and P. Kohli. Perforatedcnns: Acceleration through elimination of redundant convolutions. In *Advances in Neural Information Processing Systems*, pages 947–955, 2016.

[17] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[18] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[19] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[20] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[21] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[22] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[24] Y. He and S. Han. Adc: Automated deep compression and acceleration with reinforcement learning. *arXiv preprint arXiv:1802.03494*, 2018.

[25] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.

[26] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, page 6, 2017.

[27] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[28] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[29] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017.

[30] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[31] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[32] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[33] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[35] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[36] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[37] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[38] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[39] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[40] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[41] S. Lin, R. Ji, Y. Li, C. Deng, and X. Li. Towards compact convnets via structure-sparsity regularized filter pruning. *arXiv preprint arXiv:1901.07827*, 2019.

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

[43] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015.

[44] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763. IEEE, 2017.

[45] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[46] J.-H. Luo and J. Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*, 2018.

[47] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[48] M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

[49] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. 2016.

[50] A. Polyak and L. Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015.

[51] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[52] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[53] V. Roth and B. Fischer. The group-lasso for generalized linear models: uniqueness of solutions and efficient algorithms. In *Proceedings of the 25th international conference on Machine learning*, pages 848–855. ACM, 2008.

[54] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6655–6659. IEEE, 2013.

[55] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[56] V. Sindhwani, T. Sainath, and S. Kumar. Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems*, pages 3088–3096, 2015.

[57] P. Singh, V. K. Verma, P. Rai, and V. P. Namboodiri. Leveraging filter correlations for deep model compression. *arXiv preprint arXiv:1811.10559*, 2018.

[58] S. W. Stepniewski and A. J. Keane. Pruning backpropagation neural networks using modern stochastic optimisation techniques. *Neural Computing & Applications*, 5(2):76–98, 1997.

[59] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *arXiv preprint arXiv:1412.7580*, 2014.

[60] H. Wang, Q. Zhang, Y. Wang, and H. Hu. Structured pruning for efficient convnets via incremental regularization. *arXiv preprint arXiv:1811.08390*, 2018.

[61] Y. Wang, C. Xu, C. Xu, and D. Tao. Beyond filters: Compact feature map for portable deep model. In *International Conference on Machine Learning*, pages 3703–3711, 2017.

[62] Y. Wang, C. Xu, S. You, D. Tao, and C. Xu. Cnnpack: Packing convolutional neural networks in the frequency domain. In *Advances in neural information processing systems*, pages 253–261, 2016.

[63] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[64] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[65] J. Xue, J. Li, and Y. Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*, pages 2365–2369, 2013.

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

[67] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 184–199, 2018.

[68] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2016.

[69] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pages 662–677. Springer, 2016.

[70] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 883–894, 2018.