Article:

Manipulating Identical Filter Redundancy for Efficient Pruning on Deep and Complicated CNN

Tianxiang Hao, Xiaohan Ding, Jungong Han, Yuchen Guo, Guiguang Ding

Abstract—The existence of redundancy in Convolutional Neural Networks (CNNs) enables us to remove some filters/channels with acceptable performance drops. However, the training objective of CNNs usually tends to minimize an accuracy-related loss function without any attention paid to the redundancy, making the redundancy distribute randomly on all the filters, such that removing any of them may trigger information loss and accuracy drop, necessitating a fine-tuning step for recovery. In this paper, we propose to manipulate the redundancy during training to facilitate network pruning. To this end, we propose a novel Centripetal SGD (C-SGD) to make some filters identical, resulting in *ideal redundancy patterns*, as such filters become *purely redundant* due to their duplicates; hence removing them does not harm the network. As shown on CIFAR and ImageNet, C-SGD delivers better performance because the redundancy is better organized, compared to the existing methods. The efficiency also characterizes C-SGD because it is as fast as regular SGD, requires no fine-tuning, and can be conducted simultaneously on all the layers even in very deep CNNs. Besides, C-SGD can improve the accuracy of CNNs by first training a model with the same architecture but wider layers then squeezing it into the original width.

Index Terms—Deep Learning, Convolutional Neural Network, Model Compression, Filter Pruning, Channel Pruning.

I. INTRODUCTION

CONVOLUTIONAL Neural Networks (CNNs) have become the de facto standard for computer vision and very deep architectures are much sought-after by visual tasks, such as image recognition, due to their approaching human-level performance. As CNNs grow wider and deeper, their memory footprint, power consumption and required floating-point operations (FLOPs) have increased dramatically. In this context, CNN compression and acceleration methods have been prevalent during the past few years. This paper focuses on filter pruning, a.k.a. channel pruning [1] or network slimming [2], because of its three unique features: **1)** generic - it can handle various CNNs with no assumptions on the application field, the network architecture or the deployment platform; **2)** effective - it can significantly reduce the required FLOPs of the network, which serve as the main criterion of computational burden; **3)** complementary to other techniques - it simply produces a thinner network with no customized structure or extra operation, which is orthogonal to the other model compression and acceleration methods.

In the past few years, tremendous efforts have been devoted to filter pruning techniques. Due to the widely observed redundancy in CNNs [3]–[8], it is shown that if a CNN is pruned without a big decline in performance, a follow-up fine-tuning procedure may restore the performance to a certain degree. Some prior works [9]–[13] estimate the importance of filters by a variety of metrics, directly remove some filters and re-construct the network with the remaining ones. However, though the pruned filters are less important in some sense, they are not *purely redundant*, hence the performance will be degraded. Moreover, some recent powerful networks adopt complicated structures, like shortcut [14] and dense connection [15], 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 the important filters at different layers usually reside in different positions, such that some important filters have to be pruned due to constraints. To reduce the destructive impact of pruning, another family of methods [16]–[21] seeks to zero out some filters in advance, where group-Lasso Regularization [22] is frequently used. The rationale behind this is simple: the model undergoes less damage during pruning if the magnitudes of the pruned parameters have been reduced in advance because pruning filters is mathematically equivalent to setting all of their parameters to zero. However, such regularizations cannot literally zero out the filters but merely reduce the magnitudes to some extent (Sect. V-E), hence the pruning still damages the model and a fine-tuning process remains necessary [16]–[18].

We note that zeroing filters out can be regarded as producing a *redundancy pattern*, which we refer to as *small-norm redundancy* for convenience. As some filters become more redundant (*i.e.*, smaller in magnitude) than before but still not purely redundant, the small-norm redundancy pattern is non-ideal. In this paper, we also aim to produce some redundancy patterns in CNNs for filter pruning. However, unlike the non-ideal small-norm redundancy pattern, we seek to produce ideal patterns, where some filters are *purely redundant*, such that removing them is not harmful to the model. To this end, we intend to merge multiple filters into one, thus generating a redundancy pattern where some filters are *identical*. Meanwhile, supervised by the model's original objective function, the performance is maintained. Compared to the importance-based filter pruning methods, doing so requires no heuristic

T. Hao, G. Ding are with the School of Software, Tsinghua University, Beijing 100084, China. X. Ding is with Tencent AI Lab, Shenzhen 518054, China. J. Han is with the Computer Science Department, University of Sheffield, S1 4DP, UK. Y. Guo is with the Department of Automation, Tsinghua University, Beijing 100084, China. (e-mail: beyondhtx@gmail.com; xiaohding@gmail.com; jungonghan77@gmail.com; yuchen.w.guo@gmail.com; dinggg@tsinghua.edu.cn.)

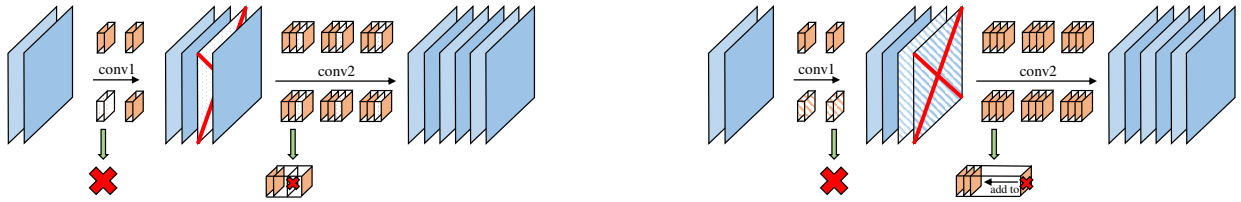Corresponding author: Yuchen Guo, Guiguang Ding.

Fig. 1: Zeroing-out *vs.* centripetal constraint. This figure shows a CNN with 4 and 6 filters at the 1st and 2nd conv layer, respectively, which takes a 2-channel input. Left: the 3rd filter at conv1 is zeroed out (*i.e.*, all the entries in the parameter tensor of the 3rd filter are close to zero, more precisely, $\boldsymbol{K}^{(1)}_{:,:,:,3} \approx \boldsymbol{0}$ using the formulation described in Sect. III-A), thus the 3rd channel is close to zero ($\boldsymbol{M}^{(1)}_{:,:,3} \approx \boldsymbol{0}$), implying that the 3rd input channels of the 6 filters at conv2 are useless. During pruning, the 3rd filter at conv1 and 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 the centripetal constraint until the 3rd and 4th channels 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. Then the 4th filter at conv1 is pruned, and the 4th input channel of every filter at conv2 is added to the 3rd channel.

knowledge about the importance of a filter. In contrast to the small-norm methods, the redundancy pattern is ideal, which enables absolutely lossless pruning and eliminates the need for a time-consuming fine-tuning process.

The motivation is an observation of the information flow in CNNs (Fig. 1). It reveals **1)** if two or more filters are trained to become identical, due to the *linearity* of convolution, we can simply retain only one filter and discard others, and add up the parameters along the corresponding input channels of the next layer. Doing so will lead to ZERO damage on performance; **2)** by encouraging multiple filters 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. Therefore, the representational capacity of our model is probably weaker than that of the original expensive model, but stronger than a counterpart with the filters being zeroed out (Sect. V-E), as the input channels corresponding to the zeroed-out filters no longer contribute to the information flow [18]. On the other hand, we will show that compared with a model without any manipulated redundancy, training with identical filters delivers higher accuracy (Sect. V-G). Our code is released at https://github.com/DingXiaoH/Centripetal-SGD to encourage further studies. Our contributions are summarized as follows:

- We propose to produce ideal redundancy patterns in CNNs by training some filters to become identical (Fig. 2) via *Centripetal SGD* (C-SGD), an efficient SGD method which can solve *constrained filter pruning*. Here "centripetal" means "several objects moving towards a center", which describes the behavior of the filters in C-SGD.
- We present an efficient implementation of C-SGD with matrix multiplications, which introduces no observable computational burden, compared to normal SGD.
- As a theoretical contribution, we show training a model with identical filters using C-SGD *from scratch* delivers higher accuracy than a counterpart without such redundancy. This serves as evidence for supporting our motivation (Fig. 1) as well as the assumption that redundancy helps the convergence of neural networks [23], [24].
- We propose a novel approach, Structural Squeezing, to
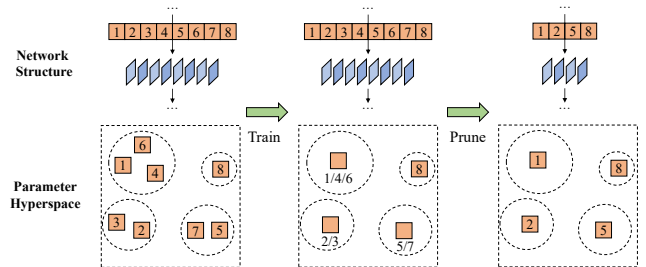
Fig. 2: The C-SGD pruning pipeline. When we seek to slim an 8-filter convolutional layer down to 4 filters, we divide the filters into 4 clusters according to their location in the parameter hyperspace. For instance, for a 3×3 convolutional layer which takes a 64-channel feature map as input, every filter kernel has $3 \times 3 \times 64 = 576$ parameters, thus the dimensionality of hyperspace is 576. During C-SGD training, the filters in each cluster become closer and eventually identical. When the training is completed, we remove all but one filter for each cluster, and adjust the following layer as Fig. 1.

improve the accuracy of CNNs, which first trains a model with wider layers and then squeezes it into the original width via C-SGD. Compared to the prior methods which fail to utilize the weights inherited from a wider model by pruning and fine-tuning, Structural Squeezing improves the performance by a clear margin.
- We do pruning experiments on CIFAR-10 and ImageNet, and earn much better performance compared with many recent competitors. Our results on COCO detection and VOC segmentation demonstrate the generalization performance of C-SGD on the downstream tasks.

II. RELATED WORK

Numerous works [25]–[33] have shown it is feasible to remove a large portion of connections (*i.e.*, weights) from a neural network without a significant performance drop. However, as such methods do not make the parameter tensors smaller but just sparser, little or no acceleration can be observed without support from specialized software and hardware platforms. In contrast, by removing filters instead of sporadic connections, we transform the wide conv layers into

narrower ones, hence the FLOPs, memory footprint and power consumption are significantly reduced. One kind of method estimates the importance of filters by some means, then selects and prunes the unimportant filters carefully to minimize performance loss. Some methods measure a filter's importance by the classification accuracy reduction (CAR) [12], [34], the Taylor-expansion criterion [11], the magnitude of conv kernels [10] and the average percentage of zero activations (APoZ) [9], respectively. Measuring the importance of some transformation form of the original conv kernel is also feasible, such as SVD decomposition [35]. Another category seeks to train the network under certain constraints to zero out some filters [16]–[21], [36], [37], where the representative is group-Lasso Regularization. In addition, the similarities between filters [31], [38] can also be used as a metric to select the redundant ones that will be pruned.

Some major drawbacks of the prior works are as follows. **1)** For the importance-based methods, the filter importance metrics are essentially heuristic, as it is not clear why the proposed metrics reflect the inherent importance of filters. Also, it is hard to judge if a heuristic metric is theoretically better than another. **2)** Since removing whole filters can degrade the performance a lot, the models are usually pruned in a layer-by-layer [9], [17] or filter-by-filter [11], [12] manner. On modern very deep CNNs, such pruning processes may not only become time-consuming but also suffer from the notorious problem of error propagation and amplification through multiple layers when estimating the filter importance [8]. **3)** Many of these works require one or more fine-tuning processes after pruning to restore the accuracy [9], [11], [17], [18], [35], [36], [38]. However, Liu *et al.* [39] have empirically found out that fine-tuning a pruned model may not always guarantee higher accuracy, compared to training from scratch, as the pruned model might be trapped into a bad local minimum. **4)** The regularization-based methods may bring a significantly higher computational burden. For example, Sect. V-E shows group-Lasso Regularization [22] slows down the training by about 2×, as it requires costly square root operations. **5)** Many of the methods cannot handle the constrained filter pruning problem on ResNets (Fig. 3), so the researchers choose to sidestep this problem by only pruning the internal layers in residual blocks [1], [2], [10]. Li *et al.* [10] and Ding *et al.* [19] tried pruning the troublesome layers according to the importance scores of others to meet the constraints, but predictably resulted in inferior accuracy. **6)** Simply grouping the filters [31] based on cosine similarities and collapsing each group into one average filter suffer from severe accuracy loss, for the reason that the filters in a normally trained CNN are actually dissimilar [31] and thus the forward processes before and after collapsing are not mathematically equivalent.

In contrast, our method features **1)** no heuristic knowledge about the filter importance, **2)** the capability of pruning every target layer simultaneously, **3)** no need for fine-tuning, **4)** negligible extra computations, **5)** global slimming on all the layers in complicated CNN architectures and **6)** equivalent collapsing transformation.

This paper represents a substantial extension of our previous conference paper [40]. The main technical novelties, compared with [40], are as follows. **1)** We propose a novel CNN training methodology, Structural Squeezing, in order to improve the performance of CNN based on C-SGD. **2)** We present more experimental results including the pruning results on VGG [41], the torchvision [42] version of ResNet-50, and the object detection and semantic segmentation results on COCO and VOC. **3)** We present more illustrations and discussions of the motivation and derivation of C-SGD together with its relation to the prior works. **4)** We present thorough comparisons and discussions of different clustering methods, including k-means, even and imbalanced clustering, on several benchmark models. **5)** We perform controlled experiments to justify the significance of solving the constrained filter pruning problem. **6)** We present more detailed discussions of the efficiency of C-SGD and its applications.

III. Filter Pruning via Centripetal SGD

A. Formulation

In modern CNNs, batch normalization [43] (BN) and linear scaling commonly follow conv layers. For simplicity and generality, we regard a conv layer together with its subsequent BN and scaling layer, if any, as a whole. Let i be the layer index, $\boldsymbol{M}^{(i)} \in \mathbb{R}^{h_i \times w_i \times c_i}$ be the output feature map of layer i with a spatial resolution of $h_i \times w_i$ and c_i channels, and $\boldsymbol{M}^{(i)}_{:,:,j}$ 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, $\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)}, \boldsymbol{\gamma}^{(i)}, \boldsymbol{\beta}^{(i)} \in \mathbb{R}^{c_i}$ and $\boldsymbol{K}^{(i)} \in \mathbb{R}^{u_i \times v_i \times c_{i-1} \times c_i}$, where $\boldsymbol{K}^{(i)}$ is the convolution kernel, $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$ are the mean and standard deviation of BN, $\boldsymbol{\gamma}^{(i)}$ and $\boldsymbol{\beta}^{(i)}$ are the scaling factor and bias term of the linear transformation, respectively. 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, a filter j at layer i refers to the five-tuple comprising all the parameter slices related to the output channel j of layer i, formally,

$$\boldsymbol{F}^{(j)} = (\boldsymbol{K}^{(i)}_{:,:,:,j}, \mu^{(i)}_j, \sigma^{(i)}_j, \gamma^{(i)}_j, \beta^{(i)}_j), \tag{1}$$

where $\boldsymbol{K}^{(i)}_{:,:,:,j}$ is the j-th slice along the axis which differentiates the c filters, *i.e.*, the 4th axis in our formulation. 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 2D convolution, the j-th output channel is

$$\boldsymbol{M}^{(i)}_{:,:,j} = \frac{\sum_{k=1}^{c_{i-1}} \boldsymbol{M}^{(i-1)}_{:,:,k} * \boldsymbol{K}^{(i)}_{:,:,k,j} - \mu^{(i)}_j}{\sigma^{(i)}_j} \gamma^{(i)}_j + \beta^{(i)}_j. \tag{2}$$

Pruning filters at a certain layer normally involves three steps: **1)** deciding which filters to prune, **2)** deleting the corresponding parameters in the kernel, *e.g.*, along the 4th axis in our formulation, and **3)** handling the vector parameters $\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta}$ accordingly. For example, the importance-based filter pruning methods [8]–[12] define the importance of filters by some means to guide the selection of filters. Let $\mathcal{I}_i$ be the filter index set of layer i (*e.g.*, $\mathcal{I}_2 = \{1, 2, 3, 4\}$ if the 2nd layer has 4 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

$$\mathcal{R}_i = \{j \in \mathcal{I}_i \mid T(\boldsymbol{F}^{(j)}) > \theta_i\}. \tag{3}$$

We construct the parameters for the slimmed layer by assembling the parameters sliced from the original tensor and vectors into the new parameters $\hat{\boldsymbol{P}}^{(i)}$. That is,

$$\hat{\boldsymbol{P}}^{(i)} = (\boldsymbol{K}^{(i)}_{:,:,:,\mathcal{R}_i}, \boldsymbol{\mu}^{(i)}_{\mathcal{R}_i}, \boldsymbol{\sigma}^{(i)}_{\mathcal{R}_i}, \boldsymbol{\gamma}^{(i)}_{\mathcal{R}_i}, \boldsymbol{\beta}^{(i)}_{\mathcal{R}_i}). \tag{4}$$

The input channels of the next layer corresponding to the pruned filters should also be discarded (Fig. 1),

$$\hat{\boldsymbol{P}}^{(i+1)} = (\boldsymbol{K}^{(i+1)}_{:,:,\mathcal{R}_i,:}, \boldsymbol{\mu}^{(i+1)}, \boldsymbol{\sigma}^{(i+1)}, \boldsymbol{\gamma}^{(i+1)}, \boldsymbol{\beta}^{(i+1)}). \tag{5}$$

Then we initialize the new network using $\hat{\boldsymbol{P}}^{(i)}$ and $\hat{\boldsymbol{P}}^{(i+1)}$.

B. Centripetal SGD

In this subsection, we present the rule of updating C-SGD together with some discussions of its properties.

For each layer, we first divide the filters into clusters, where 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 specific cluster in the form of a filter index set, respectively. We generate the clusters by k-means [44] or *arbitrarily*, between which our experiments demonstrate only minor difference (Sect. V-D). In the following sections, we use k-means clustering unless otherwise noted.

- **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. 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 remaining filters (*i.e.*, number of clusters) at layer i, respectively, each cluster will have $\lceil c_i/r_i \rceil$ filters at most. For example, if the 2nd layer has 6 filters and we wish to slim it to 4 filters, we will have $\mathcal{C}_2 = \{\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4\}$, $\mathcal{H}_1 = \{1, 2\}, \mathcal{H}_2 = \{3, 4\}, \mathcal{H}_3 = \{5\}, \mathcal{H}_4 = \{6\}$.
- **Imbalanced clustering.** An extreme solution is to put $c_i - r_i + 1$ filters into one single cluster, such that each of the other clusters has only one filter. In the above example, we will have $\mathcal{H}_1 = \{1, 2, 3\}, \mathcal{H}_2 = \{4\}, \mathcal{H}_3 = \{5\}, \mathcal{H}_4 = \{6\}$.

We use $H(j)$ to denote the cluster containing filter j, *e.g.*, $H(3) = \mathcal{H}_1$ and $H(6) = \mathcal{H}_4$ in the above example of imbalanced clustering. Let $\boldsymbol{F}^{(i)}_j(t)$ be the kernel or a vector parameter of filter j in layer i at iteration t, the update rule of C-SGD is

$$\boldsymbol{F}^{(i)}_j(t+1) = \boldsymbol{F}^{(i)}_j(t) + \tau \Delta \boldsymbol{F}^{(i)}_j(t),$$

$$\Delta \boldsymbol{F}^{(i)}_j(t) = -\frac{\sum_{k \in H(j)} \frac{\partial L(t)}{\partial \boldsymbol{F}^{(i)}_k(t)}}{|H(j)|} - \eta \boldsymbol{F}^{(i)}_j(t) \tag{6}$$

$$+ \epsilon \left(\frac{\sum_{k \in H(j)} \boldsymbol{F}^{(i)}_k(t)}{|H(j)|} - \boldsymbol{F}^{(i)}_j(t) \right),$$

where $L(t)$ is the original objective function at iteration t, τ is the learning rate, η is the original weight decay factor, and ϵ is the only hyper-parameter we introduced, the *centripetal strength*.

The intuition behind Eq. 6 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.

Let $\mathcal{L}$ be the layer index set, *e.g.*, $\mathcal{L} = \{1, 2, 3\}$ if the model has 3 convolutional layers, we use the *sum of squared kernel deviation* χ to measure the intra-cluster similarity, *i.e.*, how close filters are in each cluster,

$$\chi = \sum_{i \in \mathcal{L}} \sum_{j \in \mathcal{I}_i} \| \boldsymbol{K}^{(i)}_{:,:,:,j} - \frac{\sum_{k \in H(j)} \boldsymbol{K}^{(i)}_{:,:,:,k}}{|H(j)|} \|_2^2. \tag{7}$$

It is easy to derive from Eq. 6 that χ is lowered *monotonically* and *exponentially* with a constant learning rate τ, if the floating-point operation errors are ignored. We will give proof in the next section.

In practice, we fix η and reduce τ with time just as we do in regular SGD training. For ϵ, the performance of C-SGD is not sensitive to it. Setting ϵ without careful selection still leads to good results. Intuitively, C-SGD training with a large ϵ prefers rapid change to a stable transition. If ϵ is too large, *e.g.*, 10, the filters are merged immediately 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 negligible for 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 for absolutely lossless pruning, sooner or later. In this sense, we claim that such a redundancy pattern is ideal.

C. Properties of C-SGD

In this section, we will dive deeper into the properties of C-SGD. We start by analyzing the parameters in the same cluster.

Theorem 1. $\forall t \geq 0, i \in \mathcal{L}, j_1 \in \mathcal{I}_i, j_2 \in H(j_1)$, *we have*
$$\boldsymbol{F}^{(i)}_{j_1}(t+1) - \boldsymbol{F}^{(i)}_{j_2}(t+1) = (1 - \tau(\epsilon + \eta))(\boldsymbol{F}^{(i)}_{j_1}(t) - \boldsymbol{F}^{(i)}_{j_2}(t))$$

Proof. According to Eq. 6,

$$\boldsymbol{F}^{(i)}_{j_1}(t+1) - \boldsymbol{F}^{(i)}_{j_2}(t+1)$$
$$= \boldsymbol{F}^{(i)}_{j_1}(t) + \tau \Delta \boldsymbol{F}^{(i)}_{j_1}(t) - (\boldsymbol{F}^{(i)}_{j_2}(t) + \tau \Delta \boldsymbol{F}^{(i)}_{j_2}(t))$$
$$= (1 - \tau(\epsilon + \eta))(\boldsymbol{F}^{(i)}_{j_1}(t) - \boldsymbol{F}^{(i)}_{j_2}(t))$$
$\square$

From Theorem 1 we can derive that for two arbitrary points in the same cluster, the vector from one point to the other keeps its direction unchanged and reduces its length according to a fixed ratio $(1 - \tau(\epsilon + \eta))$ as the iteration increases. From a wider perspective, the hyper-polygon constructed by the points in the same cluster keeps its shape unchanged and merely shrinks to $(1 - \tau(\epsilon + \eta))$ the original scale every time the training goes on.

Theorem 2. χ *is lowered monotonically and exponentially, as iteration t increases.*

Proof.

$$\chi(t+1)$$

$$= \sum_{i\in\mathcal{L}} \sum_{\mathcal{H}\in\mathcal{C}_i} \sum_{j\in\mathcal{H}} \| \boldsymbol{F}_j^{(i)}(t+1) - \frac{\sum_{k\in\mathcal{H}}\boldsymbol{F}_k^{(i)}(t+1)}{|\mathcal{H}|} \|_2^2$$

$$= \sum_{i\in\mathcal{L}} \sum_{\mathcal{H}\in\mathcal{C}_i} \sum_{j\in\mathcal{H}} \| \frac{|\mathcal{H}|\boldsymbol{F}_j^{(i)}(t+1) - \sum_{k\in\mathcal{H}}\boldsymbol{F}_k^{(i)}(t+1)}{|\mathcal{H}|} \|_2^2$$

$$= \sum_{i\in\mathcal{L}} \sum_{\mathcal{H}\in\mathcal{C}_i} \sum_{j\in\mathcal{H}} \| \frac{\sum_{k\in\mathcal{H}}(\boldsymbol{F}_j^{(i)}(t+1) - \boldsymbol{F}_k^{(i)}(t+1))}{|\mathcal{H}|} \|_2^2$$

$$= (1-\tau(\epsilon+\eta))^2 \sum_{i\in\mathcal{L}} \sum_{\mathcal{H}\in\mathcal{C}_i} \sum_{j\in\mathcal{H}} \| \frac{\sum_{k\in\mathcal{H}}(\boldsymbol{F}_j^{(i)}(t) - \boldsymbol{F}_k^{(i)}(t))}{|\mathcal{H}|} \|_2^2$$

$$= (1-\tau(\epsilon+\eta))^2 \chi(t)$$

Therefore, $\{\chi(t)\}$ is a geometric progression with common ratio $(1-\tau(\epsilon+\eta))^2$. In practice, since $0 < (1-\tau(\epsilon+\eta))^2 < 1$ and $\chi(0) > 0$, χ is lowered *monotonically* and *exponentially*. □

According to Theorem 2, the update rule of C-SGD ensures that the intra-cluster distance χ rapidly decreases in the training. A good clustering method like k-means leads to a good starting of C-SGD. However, each cluster will collapse to a single point finally if the training time is long enough, whatever the cluster looks like in the beginning. Such property indicates that C-SGD is not very sensitive to the division of the initial clusters. As we will show in Sect. V-D, the pruning results of different clustering methods (k-means, even or imbalanced) are close.

D. Efficient Implementation of C-SGD

The efficiency of modern CNN training and deployment platforms is dependent on large-scale tensor operations. We, therefore, seek to implement C-SGD by efficient matrix multiplications which introduce minimal computational burden. 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. 9 and Eq. 10 such that Eq. 8 is equivalent to Eq. 6, which can be easily verified. Obviously, when the number of clusters equals that of the filters, Eq. 8 degrades into normal SGD with $\boldsymbol{\Gamma} = \text{diag}(1), \boldsymbol{\Lambda} = \text{diag}(\eta)$. 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.

$$\boldsymbol{W}(t+1) = \boldsymbol{W}(t) - \tau(\frac{\partial L(t)}{\partial \boldsymbol{W}(t)}\boldsymbol{\Gamma} + \boldsymbol{W}(t)\boldsymbol{\Lambda}). \quad (8)$$

$$\boldsymbol{\Gamma}_{m,n} = \begin{cases} \dfrac{1}{|H(m)|} & \text{if } H(m) = H(n), \\ 0 & \text{elsewise}. \end{cases} \quad (9)$$
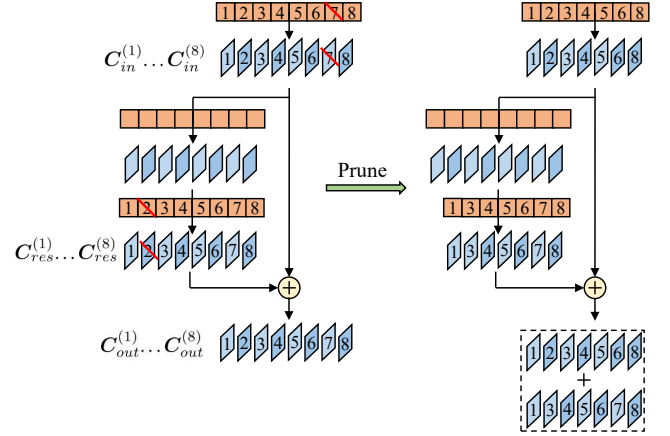

Fig. 3: We explain the problem of constrained filter pruning in a ResNet where some layers must be pruned in the same pattern as others. Here an 8-channel feature map (denoted by $\boldsymbol{C}_{in}^{(i)} \in \mathbb{R}^{h\times w}, 1\le i\le 8$) is fed into a residual block, and the output of the last layer of residual block $\boldsymbol{C}_{res}^{(i)}$ is added onto $\boldsymbol{C}_{in}^{(i)}$. Therefore, we have $\boldsymbol{C}_{out}^{(i)} = \boldsymbol{C}_{in}^{(i)} + \boldsymbol{C}_{res}^{(i)}, \forall 1\le i\le 8$. By filter pruning, we expect to remove one or several channels without breaking the correspondence between the remaining channels which are added up. Therefore, the remaining sets of the first and last layers must be the same. Otherwise, if we prune the 7th filter at the first layer and the 2nd filter at the last layer of the residual block, for example, we will end up with $\boldsymbol{C}_{out}^{(2)} = \boldsymbol{C}_{in}^{(2)} + \boldsymbol{C}_{res}^{(3)}, \boldsymbol{C}_{out}^{(6)} = \boldsymbol{C}_{in}^{(6)} + \boldsymbol{C}_{res}^{(7)} \dots$ Hence the correspondence breaks, and the model is destroyed.

$$\boldsymbol{\Lambda}_{m,n} = \begin{cases} \eta + \epsilon - \dfrac{\epsilon}{|H(m)|} & \text{if } m = n, \\ -\dfrac{\epsilon}{|H(m)|} & \text{if } m\ne n, H(m) = H(n), \\ 0 & \text{elsewise}. \end{cases} \quad (10)$$

E. Filter Trimming after C-SGD Training

After training, 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. Since the ideal redundancy patterns have emerged, *i.e.*, the filters in each cluster have become identical, such choice is unimportant and different choices lead to the same results.

$$\mathcal{R}_i = \{\min(\mathcal{H}) \mid \forall \mathcal{H}\in\mathcal{C}_i\}. \quad (11)$$

For the following layer, we add the to-be-deleted input channels to the corresponding remaining one (Fig. 1),

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

then we delete the redundant filters and the input channels of the following layer as Eq. 4, 5. Due to the linearity of conv (Eq. 2), no damage is caused, hence *no fine-tuning* is needed.

F. C-SGD for Constrained Filter Pruning

Recently, several efficient and compact CNN architectures [14], [15] have emerged. Although some works [7]–[9], [11], [45] have shown that the classical plain CNNs, *e.g.*,
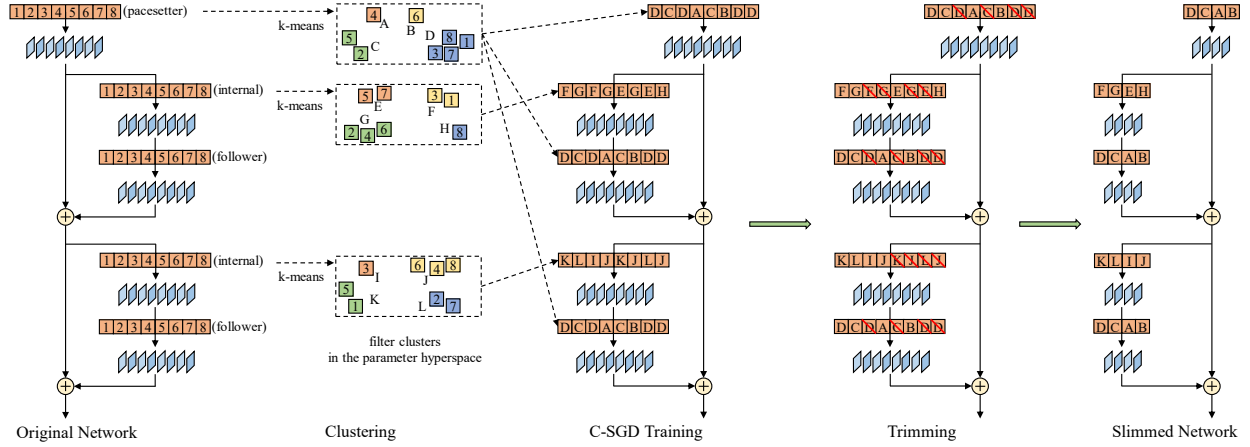
Fig. 4: Sketch for slimming ResNets. We take the first stage of a toy ResNet where every layer has 8 filters for example. Since every convolutional layer is directly followed by exactly one batch normalization (BN) layer, we view them as a whole. We generate clusters for the pacesetter and internal layers in each stage by k-means for example. Before C-SGD training, the clustering result of a pacesetter is assigned to its followers in order to produce the same redundancy pattern.
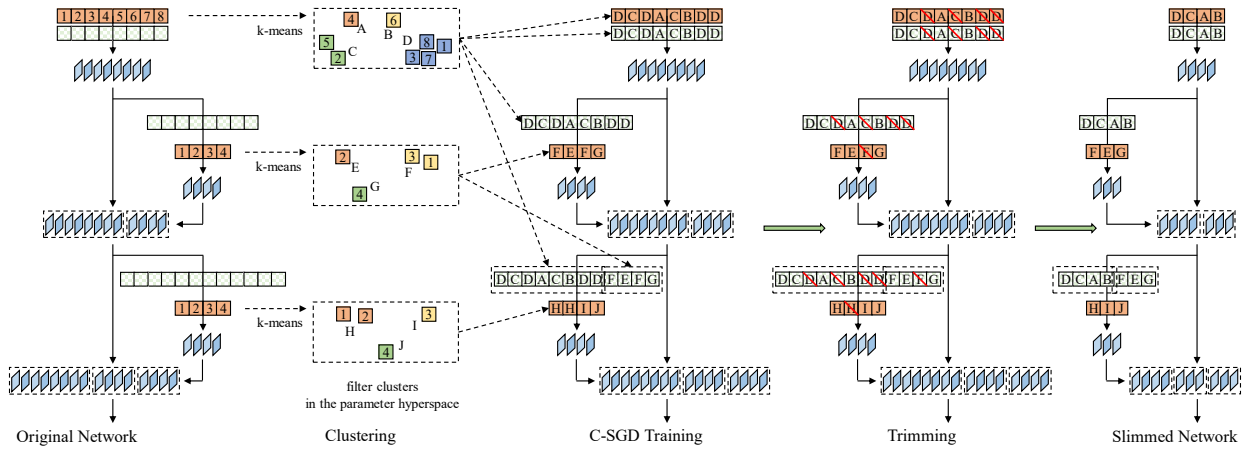

Fig. 5: Sketch for slimming DenseNets. We take a toy DenseNet with a growth rate of 4 for example. Considering the special dense connection and pre-activation structure of DenseNets, we treat the BN layers separately, which are denoted by rectangles with a chessboard-like background. As the output of every conv layer serves as the input of one or more BN layers, we generate clusters for every convolutional layer and apply the clustering results $\mathcal{C}$ to every following BN layer *at the corresponding position*, such that the gradients of γ and β are transformed as the preceding convolutional layers. Note that a BN layer can be regarded as a degraded case of the definition in Sect. III-A without loss of generality.

AlexNet [46] and VGG [41], 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. Filter pruning for very deep and complicated CNNs is challenging due to: **1)** Firstly, these models are designed in consideration of the computational efficiency, which makes them inherently compact and efficient. **2)** Secondly, these networks are significantly deeper than the classical ones, thus layer-by-layer pruning becomes too time-consuming, and the errors can increase dramatically when propagated through multiple layers, thus making the estimation of filter importance less accurate [8]. **3)** Most importantly, some innovative structures are heavily used in these networks, *e.g.*, shortcuts [14] and dense connections [15], raising an open problem of constrained filter pruning.

For example, 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 the same, or the model will be damaged so badly that fine-tuning cannot restore its accuracy. However, important filters in the pacesetters and followers usually reside in different positions, such that we have to prune some important filters in some layers due to constraints. An intuitive explanation is shown in Fig. 3.

In some prior explorations, Li *et al.* [10] sidestep this problem by only pruning the *internal* layers on ResNet-56, *i.e.*, the first layers in the residual blocks. Liu *et al.* [2] and He *et al.* [1] skip these troublesome layers and insert an extra sampler layer before the internal layers during inference time to reduce the input channels. From a holistic perspective, the networks are not literally "slimmed" but actually "clipped".

We present a solution to this open problem with 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 training, the same redundancy patterns among filters at both layer p and q are produced. *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 others with no performance loss. Fig. 4 and Fig. 5 illustrate how we prune ResNets and DenseNets, respectively, where each rectangle represents a filter, and different filters labeled by the same letter become identical during training.

IV. STRUCTURAL SQUEEZING: A NEW TRAINING METHODOLOGY WITH C-SGD

In this paper, we propose Structural Squeezing, a novel CNN training methodology based on C-SGD, to improve the performance of CNN without any extra parameters or FLOPs. Concretely, given an off-the-shelf CNN architecture, we first train a model with regular SGD from scratch, where some layers are wider than the original. Naturally, such a wide model will be more powerful than the original one but at the cost of more parameters and computations. Then we use C-SGD to slim it down to the original structure. As will be shown in Sect. V-I, the performance of the resulting model will be lower than the wide one, but higher than a counterpart with the same structure trained with regular SGD. Intuitively, when the filters in each cluster are constrained to grow closer, the learned knowledge is gradually "squeezed" into the cluster center, *i.e.*, the merged filter, such that the resulting model becomes more powerful than the normal counterpart. Interestingly, it is observed that scaling and pruning globally, including those troublesome layers, yields better performance than only scaling and pruning the easy-to-prune layers. This observation highlights the significance of C-SGD in solving the constrained filter pruning problem.

The key distinguishing Structural Squeezing from simply pruning a bigger model into a smaller one with the traditional pruning methods is that the former improves the performance by a significant margin while the latter cannot outperform a regularly trained counterpart due to the weakness of their non-ideal redundancy patterns. Recently, Liu *et al*. [39] validated several pruning-and-fine-tuning methods [1], [2], [10], [11], [29], [47] and empirically found out that with the same width, the network obtained by pruning delivers no better performance than a counterpart trained from scratch. The authors state that though the remaining weights are considered *important* by the pruning criteria, inheriting them does not help the fine-tuning process achieve better accuracy, but might trap the pruned model into a bad local minimum. However, our method does not judge the parameters by their importance and discard the unimportant ones, nor fine-tune a model after lossy pruning. On the contrary, by averaging the gradients of filters in each cluster (Eq. 6), we fully utilize the information encoded in the objective function to supervise the whole cluster and reduce the possibility of being trapped into a local minimum.

Note that Structural Squeezing is complementary to the other techniques for improving CNN performance like stronger data augmentation, advanced loss functions, *etc*. The fact that Structural Squeezing increases the training costs does not hinder its practical application because we usually care about the inference-time performance and efficiency more than the training costs in real-world applications, as we commonly train models on powerful workstations and deploy them to multiple front-end devices where the efficiency matters. By Structural Squeezing, we obtain a model of the same structure as a normally trained counterpart with better performance.

V. EXPERIMENTS

We performed several experiments to evaluate C-SGD. **1)** We validated the effectiveness of C-SGD by pruning several common benchmark models on CIFAR-10, ImageNet, COCO detection and VOC segmentation. **2)** We compared different clustering methods and discovered a minor difference. **3)** We demonstrated the superiority of C-SGD over the zeroing-out methods in the sense that C-SGD converges faster and enables lossless pruning by producing ideal redundancy patterns. **4)** A series of controlled experiments were conducted to fairly compare C-SGD and some other pruning methods with the same training settings. **5)** We found out that when both were trained from scratch, a model with identical filters outperforms another one without, thus providing empirical evidence supporting the assumption that redundancy can help the convergence of the neural network. **6)** We justified the significance of solving the constrained filter pruning problem by showing that global slimming on ResNet yields better performance than simply clipping the easy-to-prune layers. **7)** We verified the effectiveness of Structural Squeezing by training a model with the same architecture but wider layers, squeezing it into the original width and comparing it with the normally trained counterparts.

A. Pruning Results on CIFAR-10

We first evaluate C-SGD on CIFAR-10 (Table. I). Since our base models deliver different accuracy than the competitors, we present the absolute and relative error increase as the metrics to compare the change of accuracy on different base models. For example, the Top1 accuracy is 93.53% for our base VGG [41] model and 93.59% for the result labeled as C-SGD-VGG-C, such that the absolute and relative error increase are $93.53\% - 93.59\% = -0.06\%$ and $\frac{-0.06}{100-93.53} = -0.92\%$, respectively. For each trial we start from a well-trained base model, cluster the filters by k-means, apply C-SGD training on all the layers *simultaneously*, prune every layer and test the resulting model. The base models are trained from scratch for 600 epochs with the standard data augmentation techniques [14]: padding to 40×40, random cropping and flipping. We perform C-SGD training for 600 epochs with batch size 64 and a learning rate initialized as 3×10^{-2} then multiplied by 0.1 after 200 and 400 epochs, respectively. The hyper-parameter ϵ is casually set to 3×10^{-3}.

We start with VGG, a 13-layer plain network. As a common practice, a BN follows every conv layer. To compare with

TABLE I: Pruning Results on CIFAR-10 sorted by the FLOPs reduction ratio. Note that a negative error increase denotes an improvement in accuracy. For ResNets, "internal" and "sampler" denote that the architecture is still 16-32-64, but the internal layers of residual blocks are clipped, or the sampler layers are inserted in front of the blocks.

Model	Result	Base Top1	Pruned Top1	Top1 Error Abs/Rel ↑%	FLOPs ↓%	Params ↓%	Architecture
VGG	Li *et al.* [10]	93.25	93.40	-0.15 / -2.22	34.2	64.0	-
VGG	EigenDamage [48]	93.44	93.40	0.04 / 0.61	45.53	85.83	-
VGG	Network Slimming [2]	93.66	93.80	-0.14 / -2.20	51.0	88.5	-
VGG	**C-SGD-VGG-A**	**93.53**	**94.10**	**-0.57 / -8.80**	**61.69**	**86.28**	**-**
VGG	Jiang *et al.* [49]	93.46	93.40	0.06 / 0.91	67.6	92.7	-
VGG	**C-SGD-VGG-B**	**93.53**	**93.78**	**-0.25 / -3.86**	**75.15**	**90.09**	**-**
VGG	AOFP [50]	93.38	93.28	0.10 / 1.51	75.27	-	-
VGG	Huang *et al.* [51]	92.77	89.37	3.40 / 47.02	80.6	92.8	-
VGG	AFP [19]	92.92	92.44	0.48 / 6.77	81.39	93.51	-
VGG	Singh *et al.* [52]	93.49	93.02	0.47 / 7.21	83.43	95.83	-
VGG	**C-SGD-VGG-C**	**93.53**	**93.59**	**-0.06 / -0.92**	**85.02**	**96.54**	**-**
Res56	Channel Pruning [1]	92.8	91.8	1.0 / 13.88	50	-	sampler
Res56	ADC [53]	92.8	91.9	0.9 / 12.5	50	-	sampler
Res56	FPGM [54]	93.59	93.26	0.33 / 5.14	52.6	-	-
Res56	AFP [19]	93.93	92.94	0.99 / 16.30	60.85	60.90	10-20-40
Res56	**C-SGD-Res56-10-20-40**	**93.39**	**93.62**	**-0.23 / -3.47**	**60.85**	**60.90**	**10-20-40**
Res110	NISP-110 [8]	-	-	0.18 / -	43.78	43.25	-
Res110	GAL-0.5 [55]	93.50	92.74	0.76 / 11.6	48.5	-	-
Res110	HRank [56]	93.50	93.36	0.14 / 2.15	58.2	-	-
Res110	**C-SGD-Res110-10-20-40**	**94.38**	**94.41**	**-0.03 / -0.53**	**60.89**	**60.92**	**10-20-40**
Res164	Network Slimming [2]	94.58	94.73	-0.15 / -2.76	44.90	35.2	sampler
Res164	**C-SGD-Res164-12-24-46**	**94.83**	**95.08**	**-0.25 / -4.83**	**45.24**	**54.75**	**12-24-46**
Dense40	Network Slimming [2]	93.89	94.35	-0.46 / -7.52	55.00	65.2	-
Dense40	**C-SGD-Dense40-5-8-10**	**93.81**	**94.56**	**-0.75 / 12.11**	**60.05**	**36.16**	**5-8-10**

other competitors with different pruning ratios, we prune the base model with three target width settings labeled from A to C to reach around 60%, 75%, and 85% FLOPs reduction, respectively. As we do not intend to push the state-of-the-art results on such a simple network, we set the target width casually. *E.g.*, the 13 layers of the model labeled as C-SGD-VGG-C has 20, 50, 80, 80, 80, 80, 80, 60, 60, 60, 60, 60, 60 filters, respectively, such that around 85% FLOPs are reduced. The comparison shows the superiority of C-SGD. Though some prior works succeeded in pruning VGG with improved performance, they did not achieve an increase so significant at such high pruning ratios (*e.g.*, 0.25% at 75.15%). Moreover, we observe no accuracy drop even when the FLOPs reduction reaches 85%.

For ResNets, we aim to reduce around 60% FLOPs of every model by pruning $3/8$ of *every* conv layer, thus the parameters and FLOPs are reduced by $1 - (5/8)^2 = 61\%$. As the original ResNets have 16, 32 and 64 filters at each layer in the three stages, respectively, we denote their structure as 16-32-64, and our pruned models as 10-20-40. Aggressive as the pruning is, we observe no obvious accuracy drop. Better still, as the depth of ResNet increases, the effectiveness of C-SGD does not degrade, which distinguishes C-SGD from the layer-by-layer methods. We also produced a ResNet-164 labeled as 12-24-46, such that its FLOPs are comparable with Liu *et al.* [2].

The original DenseNet-40 has 12 filters at every incremental conv layer, while the pruned model has 5, 8 and 10 filters for the three stages, respectively, such that the FLOPs are reduced by 60.05%, and an accuracy increase is achieved, which is

consistent with but better than that of Liu *et al.* [2].

B. Pruning Results on ImageNet

Table. II shows the results on the original ResNet-50 [14], which is a common benchmark in the filter pruning literature. Since many competitors experimented with the torchvision [42] version of ResNet-50 (denoted by Res50B), we also prune it for fair comparison (Table. III). The only difference between the original ResNet-50 and Res50B is that the former conducts downsampling by the 1×1 conv at the beginning of a stage while the latter uses the 3×3 conv. For pruning each model, we train with C-SGD for 70 epochs with a learning rate initialized as 0.03 and multiplied by 0.1 at the 30th, 50th and 60th epochs, respectively. We use a batch size of 256 on 8 GPUs, weight decay of 10^{-4}, centripetal strength $\epsilon = 0.05$.

We present the comparison of C-SGD and some recent competitors using the standard data augmentation methods including bounding box distortions and color shift (Table. II). Our base model reaches a Top1/Top5 accuracy of 75.33%/92.56%. Though the base models of some competitors have different accuracies, the results are still comparable in terms of the absolute and relative error increase. Following ThiNet and Lin *et al.* [21], we slim the internal layers down to 70%, 60% and 50% of the original width, respectively.

For Res50B, we use the official pre-trained model and the default data preprocessing [61] (Table. III). Our pruned models exhibit fewer FLOPs and lower error increases. Of note is that, instead of carefully tuning the target network width, we simply apply the same pruning ratio for each internal layer. In other

TABLE II: Pruning ResNet-50 on ImageNet using bounding box distortions and color shift, sorted by the FLOPs reduction.

Result	Base Top1	Base Top5	Pruned Top1	Pruned Top5	Top1 Error Abs/Rel ↑%	Top5 Error Abs/Rel ↑%	FLOPs ↓%	Params ↓%
C-SGD-Res50-70	**75.33**	**92.56**	**75.27**	**92.46**	**0.06 / 0.24**	**0.10 / 1.34**	**36.75**	**33.38**
NISP [8]	-	-	-	-	0.89 / -	- / -	44.01	43.82
Singh *et al.* [52]	-	92.65	-	92.2	- / -	0.45 / 6.13	44.45	40.92
C-SGD-Res50-60	**75.33**	**92.56**	**74.93**	**92.27**	**0.40 / 1.62**	**0.29 / 3.89**	**46.24**	**42.83**
CFP [57]	75.3	92.2	73.4	91.4	1.9 / 7.69	0.8 / 10.25	49.6	-
Channel Pr [1]	-	92.2	-	90.8	- / -	1.4 / 17.94	50	-
HP [58]	76.01	92.93	74.87	92.43	1.14 / 4.75	0.50 / 7.07	50	32.5
ELR [59]	-	92.2	-	91.2	- / -	1 / 12.82	50	-
SSR-L2 [21]	75.12	92.30	71.47	90.19	3.65 / 14.67	2.11 / 27.40	55.76	51.56
C-SGD-Res50-50	**75.33**	**92.56**	**74.54**	**92.09**	**0.79 / 3.20**	**0.47 / 6.31**	**55.76**	**51.50**
ThiNet [60]	75.30	92.20	72.03	90.99	3.27 / 13.23	1.21 / 15.51	55.83	-

TABLE III: Pruning the standard torchvision ResNet-50 (denoted by Res50B) on ImageNet using default data augmentation.

Result	Base Top1	Base Top5	Pruned Top1	Pruned Top5	Top1 Error Abs/Rel ↑%	Top5 Error Abs/Rel ↑%	FLOPs ↓%	Params ↓%
C-SGD-Res50B-70	**76.15**	**92.87**	**75.94**	**92.88**	**0.21 / 0.88**	**-0.01 / -0.14**	**36.38**	**33.38**
GAL-0.5 [55]	76.15	92.87	71.95	90.94	4.20 / 17.61	1.93 / 27.06	43.03	-
HRank [56]	76.15	92.87	74.98	92.33	1.17 / 4.90	0.54 / 7.57	43.76	-
C-SGD-Res50B-60	**76.15**	**92.87**	**75.80**	**92.65**	**0.35 / 1.46**	**0.22 / 3.08**	**46.51**	**42.83**
FPGM [54]	76.15	92.87	74.83	92.32	1.32 / 5.53	0.55 / 7.71	53.5	-
C-SGD-Res50B-50	**76.15**	**92.87**	**75.29**	**92.39**	**0.86 / 3.60**	**0.48 / 6.73**	**55.44**	**51.50**

TABLE IV: Pruning DenseNet-121 on ImageNet.

Result	Base Top1	Base Top5	Pruned Top1	Pruned Top5	Top1 Error Abs/Rel ↑%	Top5 Error Abs/Rel ↑%	FLOPs ↓%	Params ↓%
C-SGD-Dense121-A	74.47	92.14	74.25	91.76	0.22 / 0.86	0.38 / 4.83	34.65	21.53
C-SGD-Dense121-B	74.47	92.14	73.73	91.55	0.74 / 2.89	0.59 / 7.50	42.28	29.89

words, if more layer sensitivity analyzing experiments [1], [8], [10] are conducted and the target network structures are tuned accordingly, we may well get better results.

On DenseNet-121 [15] (Table. IV), without consideration of the layers' sensitivity or the filters' importance, the same pruning ratio is applied for each stage. For C-SGD-Dense121-A, the internal layers, *i.e.*, the first layers in each dense block, are shrunk to 7/8 of the original width, and the incremental factors of the first three stages become 18, 20, 24, respectively. For C-SGD-Dense121-B, the internal layers are slimmed down to 3/4 of the original width. We prune the lower-level layers harder than the higher-level ones not because of any prior knowledge, but simply because such layers operate on higher-resolution feature maps so that reducing their width results in higher acceleration. Though DenseNet-121 is not usually chosen as a benchmark model for pruning because of its complicated and compact structure, we can slim it with a minor decrease in accuracy. Such success shows the significance of C-SGD in solving the constrained filter pruning problem and the effectiveness of our strategy (Fig. 5).

C. Semantic Segmentation and Object Detection

We verify the effectiveness of C-SGD on the downstream tasks including semantic segmentation and object detection.

First, we use the augmented VOC 2012 dataset for semantic segmentation as a common practice [62], [63], which has 10,582 images for training (*trainaug* set) and 1449 images for validation (*val* set). We construct a PSPNet [64] with the original pre-trained ResNet-50B as the backbone and fine-tune

TABLE V: Semantic segmentation results on VOC2012 and object detection results on COCO with the original and pruned ResNet-50B backbones.

Backbone	Top-1 acc on ImageNet	mIoU on VOC	AP on COCO
Original ResNet-50B	76.15	76.29	33.2
C-SGD-Res50B-70	75.94	76.36	32.8
C-SGD-Res50B-60	75.80	75.78	33.1

with a poly learning rate policy with the base of 0.01 and power of 0.9, weight decay of 10^{-4} and a global batch size of 16 on 4 GPUs for 50 epochs. Then we use the pruned models denoted as C-SGD-Res50B-70 and C-SGD-Res50B-60 (Table. III) as the backbones and fine-tune with identical settings.

Then we experiment with COCO detection. More specifically, the training set is *COCO2017train* and the validation set is *COCO2017val*. We construct a Faster RCNN [65] with FPN [66] and the original pre-trained ResNet-50B as the backbone. We fine-tune for 12 epochs with a learning rate initialized as 0.02 and multiplied by 0.1 at the 8th and 11th epochs respectively. Then we use C-SGD-Res50B-70 and C-SGD-Res50B-60 as the backbones with identical settings.

Table. V demonstrates the generalization performance of the pruned models, which show very minor or even no decrease in the mIoU on VOC and AP on COCO.

D. Studies on the Clustering Methods

To study the effects of different clustering methods, we experiment with the same settings as before except for even or imbalanced clustering. Table. VI shows that k-means

TABLE VI: Pruning results with k-means, even, or imbalanced clustering.

Dataset	Model	Result	Base Top1	Pruned Top1 K-means Clustering	Pruned Top1 Even Clustering	Pruned Top1 Imbalanced Clustering
CIFAR-10	VGG	C-SGD-VGG-C	93.53	**93.59**	93.25	93.19
CIFAR-10	ResNet-56	C-SGD-Res56-10-20-40	93.39	**93.62**	93.44	93.45
CIFAR-10	DenseNet-40	C-SGD-Dense40-5-8-10	93.81	**94.56**	94.37	93.94
ImageNet	ResNet-50	C-SGD-Res50-70	75.33	**75.27**	75.14	74.93

outperforms the other two clustering methods by a narrow margin, due to the lower intra-cluster distance in the parameter hyperspace. Interestingly, our experiments indicate that the effectiveness of C-SGD-based pruning does not significantly depend on the quality of filter clusters $\mathcal{C}$, since reasonable performance can be achieved with arbitrarily generated clusters.

E. Making Filters Identical vs. Zeroing Filters Out

As making filters identical and zeroing filters out [7], [16]–[19], [21] 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 network trained with C-SGD and the one with group-Lasso Regularization [22]. For C-SGD, the number of clusters at each layer is 5/8 of the number of filters. For Lasso, 3/8 of the original filters in the pacesetters and the 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 (Eq. 7) 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,

$$\phi = \sum_{i \in \mathcal{L}} \sum_{j \in \mathcal{P}_i} \| \boldsymbol{K}^{(i)}_{:,:,:,j} \|_2^2 . \qquad (12)$$

Fig. 6 shows the curves of χ, ϕ and 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} (2×10^6 times smaller than the initial value), pruning still causes obvious damage (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 a smaller learning rate or stronger group-Lasso Regularization are conducted. We reckon this is due to the error propagation and accumulation in very deep CNNs [8]. **2)** By C-SGD, χ is reduced *monotonically* and perfectly *exponentially*, which suggests faster convergence. In other words, the filters in each cluster can become *infinitely close* to each other at a *constant rate* with a constant learning rate. In the early stage of training, the filters have not become close enough such that pruning degrades the performance (seen from the difference between "C-SGD before pruning" and "C-SGD after pruning" during the beginning 100 epochs). But after 100 epochs, the pruning
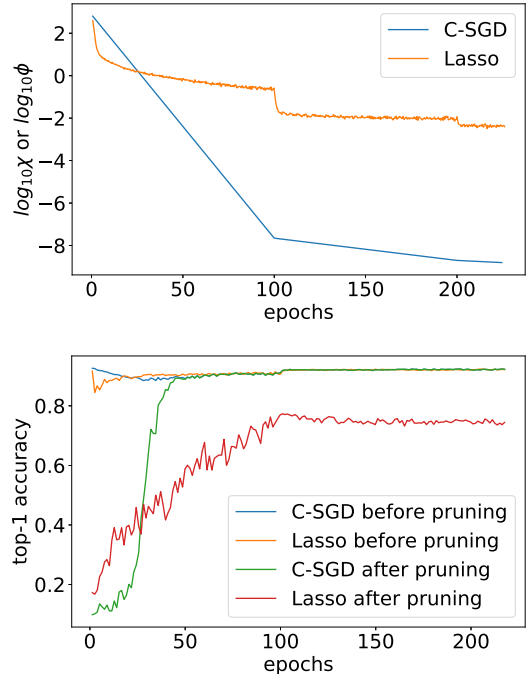

Fig. 6: Training process with C-SGD or group-Lasso on ResNet-56. Note the logarithmic scale of the upper figure. Left: values of χ or ϕ. Right: validation accuracy.

causes *absolutely no* damage. **3)** Training with group-Lasso is $2\times$ slower than C-SGD due to its computational intensity.

F. C-SGD vs. Other Filter Pruning Methods

In this subsection, we compare C-SGD with other filter pruning methods through a series of controlled experiments on DenseNet-40 [15]. We slim *every* incremental convolutional layer of a well-trained DenseNet-40 to 3 and 6 filters, respectively. The experiments are repeated 3 times, and the mean $\pm$ std curves are presented in Fig. 7. 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 guarantee the complete convergence of every competitor to ensure the fairness of comparison. For our method, the models are trained with C-SGD and trimmed. For Magnitude- [10], APoZ- [9] and Taylor-expansion-based [11], the models are pruned by the different criteria and fine-tuned. The models labeled as Lasso are trained with group-Lasso Regularization for 600 epochs in advance, pruned, then fine-tuned for *another* 600 epochs with the same learning rate schedule, such 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), and the collected results reveal the superiority of C-SGD in terms of higher accuracy and also better stability. Especially, though group-Lasso Regularization can indeed reduce the performance drop

(a) Three filters per layer.
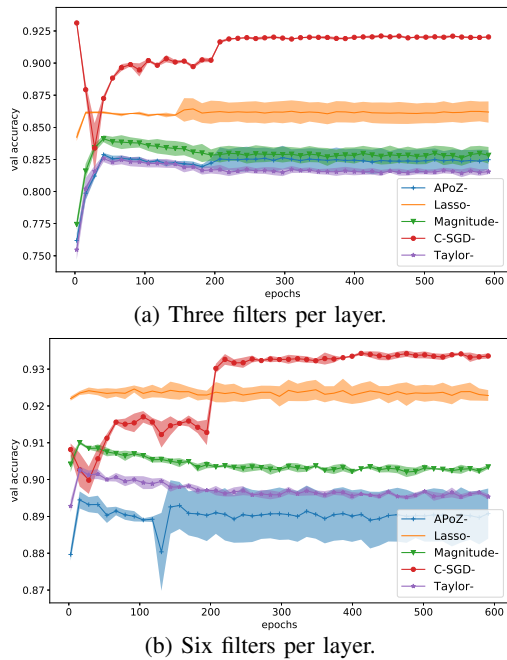

(b) Six filters per layer.

Fig. 7: Controlled pruning experiments on DenseNet-40

TABLE VII: Accuracy of scratch-trained DenseNet-40, VGG, ResNets using normal SGD or C-SGD with identical filters.

Dataset	Model	Normal SGD	C-SGD
CIFAR-10	DenseNet-3	88.60	**89.96**
CIFAR-10	VGG-1/2	92.49	**93.22**
CIFAR-10	ResNet-56-10-20-40	91.78	**92.81**
ImageNet	ResNet-50-30	69.67	**72.54**

caused by pruning, it is outperformed by C-SGD by a large margin. Another interesting observation is that the models pruned by the importance metrics are unstable and trapped into a bad local minimum, which is consistent with [39], as the accuracy curves increase steeply in the beginning but slightly decline afterward.

These observations suggest it is better to train a wide network and equivalently transform it into a narrower one than to fine-tune it after pruning, which is consistent with prior works [23], [24] that highlighted the redundancy was necessary for overcoming a non-convex optimization problem.

G. *Redundant Training* vs. *Normal Training*

We verify the significance of training with manipulated redundant filters. However, we need to eliminate the effects of the well-trained base model, or we cannot tell whether the difference in the final accuracy is due to the redundancy during the training process or the powerful weights of the well-trained big model. Concretely, we train a narrow CNN with normal SGD and compare it with another model trained using C-SGD with the equivalent width *from scratch*. To be specific, after random initialization, the latter produces some identical filters during C-SGD training and will have the same width as the former after pruning. For example, if a model has $2\times$ number of filters as the normal counterpart but every two filters are identical, they will end up with the same structure.

TABLE VIII: Top1 accuracy of ResNet-56 pruned by global slimming or clipping the internal layers.

	Resulting width	Top1 acc	FLOPs ↓
Global slimming	[10,10]-[20,20]-[40,40]	**93.62**	60.85
Clipping	[6,16]-[12,32]-[24,64]	91.77	61.76

On DenseNet-40, we evenly divide the 12 filters at each incremental conv layer into 3 clusters, use C-SGD to train the network from scratch, then trim it into a model with 3 filters per incremental layer. *I.e.*, every 4 filters grow centripetally. In contrast, we train a DenseNet-40 with originally 3 filters per layer with normal SGD. After that, we experiment on VGG, slimming each layer to 1/2 of the original width. We also experiment on ResNet-56 with a target structure of 10-20-40 and on ResNet-50 where every internal layer is reduced to 30% of the original. Table. VII shows that the redundant filters do help, compared to a normally trained counterpart with the equivalent width. This observation supports our intuition and assumption that the centripetally growing filters can enhance the model's representational capacity because though these filters are constrained, their corresponding input channels of the succeeding layers are still in full use and can grow without constraints in the parameter hyperspace (Fig. 1).

In other words, though C-SGD is originally designed for filter pruning on an off-the-shelf model, in some cases when a well-trained model is unavailable, we can use C-SGD to train a wide model from scratch and trim it into the desired structure. Though doing so delivers a lower accuracy than pruning a well-trained model (e.g., 92.81 *vs.* 93.62 on ResNet-56-10-20-40, Table. VII and I), we can still obtain a more powerful model than training from scratch using normal SGD.

H. *Global Slimming* vs. *Clipping Some Layers*

We show that with the same target FLOPs, global "slimming" yields better results than simply "clipping" some of the layers. Concretely, we prune a ResNet-56 on CIFAR-10 to reach a comparable level of FLOPs as C-SGD-Res56-10-20-40 (Table. I). Instead of slimming every layer to 5/8 of the original width, we use C-SGD to prune the internal layers only, *i.e.*, the first layers in each residual block. To realize 60% FLOPs reduction, we slim such layers to 3/8 of the original width. We use $[x, y]$ to denote the structure of a ResNet stage where the first layer in every residual block has x filters and the second has y. Table. VIII shows clipping the internal layers delivers a significantly lower accuracy, which demonstrates the superiority of global slimming over simply clipping some layers, given a specific overall pruning ratio.

I. *Structural Squeezing for More Powerful CNNs*

Structural Squeezing is a methodology to improve CNN based on C-SGD. The resulting model will deliver a higher level of accuracy with the same computational budgets as a normally trained counterpart. Concretely, we choose a mature CNN as the baseline, train a network with the same architecture but wider layers from scratch using regular SGD, and then use C-SGD to squeeze it into the original width.

TABLE IX: Structural Squeezing on VGG and ResNet-50. Of note is that we calculate the required FLOPs of every model in the same manner as Luo *et al*. [47], [60], such that the FLOPs are $2\times$ as those reported in other papers [10], [14].

Dataset	Model	Result	Top1	FLOPs	Layer Width
CIFAR-10	VGG	Baseline	93.53	626M	64-128-256-512
CIFAR-10	VGG	$2\times$ scaled	93.69	2499M	128-256-512-1024
CIFAR-10	VGG	$2\times$ pruned	**93.97**	626M	64-128-256-512
ImageNet	ResNet-50	Baseline	75.33	7.71B	64-[64-64-256]-[128-128-512]-[256-256-1024]-[512,512,2048]
ImageNet	ResNet-50	Global $1.25\times$	76.97	11.98B	80-[80-80-320]-[160-160-640]-[320-320-1280]-[640,640,2560]
ImageNet	ResNet-50	Global $1.25\times$ pruned	**76.23**	7.71B	64-[64-64-256]-[128-128-512]-[256-256-1024]-[512,512,2048]
ImageNet	ResNet-50	Bottleneck $2\times$	76.82	13.05B	64-[64-128-256]-[128-256-512]-[256-512-1024]-[512,1024,2048]
ImageNet	ResNet-50	Bottleneck $2\times$ pruned	75.88	7.71B	64-[64-64-256]-[128-128-512]-[256-256-1024]-[512,512,2048]

On CIFAR-10, a $2\times$ scaled VGG is trained from scratch with normal SGD, *i.e.*, every layer of the model is $2\times$ as wide as the normal VGG architecture. We then slim it down to the original structure by pruning half of the filters at each layer. On ImageNet, we train a $1.25\times$ scaled ResNet-50 from scratch, and slim it down to the original structure. Note that *every* conv layer is widened to $1.25\times$ of its original width, including the pacesetters and followers, which are considered troublesome by the prior works. We then use C-SGD to prune every layer simultaneously. We also experiment with another model scaled differently, where only the bottleneck layers (*i.e.*, the internal 3×3 layers in residual blocks) are scaled by $2\times$.

Table. IX shows the pruned models consistently beat the counterparts trained with regular SGD by a clear margin. Intuitively, when the filters in each cluster are constrained to grow closer, the learned knowledge is gradually "squeezed" into the cluster center, *i.e.*, the merged filters, such that the resulting model becomes more powerful than the normal counterpart. Interestingly, Global $1.25\times$ pruned outperforms Bottleneck $2\times$ pruned (76.25 *vs*. 75.88 Top1 accuracy, 0.74 *vs*. 0.94 error increase), though Bottleneck $2\times$ requires more computations. It suggests scaling and squeezing globally, including those troublesome layers, yields better performance. This observation again highlights the significance of C-SGD in solving the constrained filter pruning problem together with the results shown in Sect. V-H.

VI. DISCUSSIONS ON THE EFFICIENCY

The total time required for pruning is determined by the training (plus fine-tuning, if any) epochs, the training speed, the time consumed by the other algorithms (for those non-end-to-end methods), and the pruning granularity (*i.e.*, the number of layers/filters to prune at a time). C-SGD is efficient because it is end-to-end, requires no fine-tuning, runs as fast as regular SGD and prunes all the layers simultaneously.

C-SGD requires no fine-tuning after pruning. Sect. V-E shows the filters in each cluster become *infinitely close* to each other at a *constant rate* with a constant learning rate. This property shows the superiority of the identical-filter redundancy pattern over the small-norm pattern, as the latter cannot zero out the filters, but only reduce the magnitude of their parameters. As trimming the identical filters causes no performance drop, there is no need for a fine-tuning process, which is essential in many prior works [1], [2], [8]–[13], [17], [18], [47], [53]. **C-SGD allows one-step pruning on very deep CNNs.** The effectiveness and efficiency of C-SGD on very deep CNNs distinguish C-SGD from the

layer-by-layer [1], [9], [17], [47], [53] or filter-by-filter [11], [12] pruning methods. Many prior works choose to prune layer by layer because pruning too many layers at once may damage the network so severely that it cannot be fine-tuned to reach a satisfactory level of accuracy. In addition, the relative importance of filters is usually affected by the subsequent layers [8], such that pruning several layers stacked together at once may lead to poor estimation of the importance of filters. In contrast, C-SGD can produce the desired redundancy patterns on all the layers simultaneously to prune them all at once. In practice, we observe no accuracy drop caused by the trimming step, even in very deep CNNs like ResNet-164 and DenseNet-121. **C-SGD introduces negligible extra computational burden.** We construct the averaging matrix Γ and decaying matrix Λ according to the clustering results $\mathcal{C}$ as two constants, and store them in the GPU memory. Compared to normal SGD, for each kernel tensor at each training iteration, the only extra computations introduced are two matrix multiplications (Eq. 8), which consume minimal extra time and energy. In practice, the difference in the training speed between C-SGD and normal SGD is not observed. On the same machine with 8 GPUs, normal SGD processes 2298 images/s, and C-SGD processes 2273 images/s, for the training of the same ResNet-50. The relative difference is merely 1%. In contrast, group-Lasso slows down the training significantly, as it requires costly square root operations.

VII. CONCLUSION

We proposed to manipulate redundancy patterns by making some filters identical for pruning. By C-SGD, we have **1)** solved an open problem of constrained filter pruning on very deep CNNs with complicated architectures, **2)** beaten many recent competitors on common benchmarks under comparable FLOPs reduction ratio, **3)** presented empirical evidence for the assumption that redundancy facilitates training, which may encourage future studies, and **4)** proposed Structural Squeezing, a methodology to improve CNNs.

REFERENCES

[1] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *International Conference on Computer Vision (ICCV)*, vol. 2, 2017, p. 6.

[2] 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)*. IEEE, 2017, pp. 2755–2763.

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

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

[5] 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*, 2015, pp. 2857–2865.

[6] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations*, 2016.

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

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

[9] 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.

[10] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *5th International Conference on Learning Representations*, 2017.

[11] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *5th International Conference on Learning Representations*, 2017.

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

[13] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.

[14] 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*, 2016, pp. 770–778.

[15] 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*, 2017, pp. 2261–2269.

[16] 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*, 2015, pp. 806–814.

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

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

[19] 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, pp. 6797–6804.

[20] H. Wang, Q. Zhang, Y. Wang, L. Yu, and H. Hu, "Structured pruning for efficient convnets via incremental regularization," in *International Joint Conference on Neural Networks*, 2019, pp. 1–8.

[21] 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.

[22] 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*. ACM, 2008, pp. 848–855.

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

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

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

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

[27] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *IEEE transactions on Neural networks*, vol. 8, no. 3, pp. 519–531, 1997.

[28] S. W. Stepniewski and A. J. Keane, "Pruning backpropagation neural networks using modern stochastic optimisation techniques," *Neural Computing & Applications*, vol. 5, no. 2, pp. 76–98, 1997.

[29] 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*, 2015, pp. 1135–1143.

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

[31] A. RoyChowdhury, P. Sharma, E. Learned-Miller, and A. Roy, "Reducing duplicate filters in deep neural networks," in *NIPS workshop on Deep Learning: Bridging Theory and Practice*, vol. 1, 2017, p. 1.

[32] 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)*, 2018, pp. 184–199.

[33] S. Vadera and S. Ameen, "Methods for pruning deep neural networks," *IEEE Access*, vol. 10, pp. 63 280–63 300, 2022.

[34] Z. Chen, T.-B. Xu, C. Du, C.-L. Liu, and H. He, "Dynamical channel pruning by conditional accuracy change for deep neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 799–813, 2021.

[35] M. Lin, L. Cao, S. Li, Q. Ye, Y. Tian, J. Liu, Q. Tian, and R. Ji, "Filter sketch for network pruning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 7091–7100, 2022.

[36] H. Salehinejad and S. Valaee, "Edropout: Energy-based dropout and pruning of deep neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 10, pp. 5279–5292, 2022.

[37] T. Zhang, S. Ye, X. Feng, X. Ma, K. Zhang, Z. Li, J. Tang, S. Liu, X. Lin, Y. Liu, M. Fardad, and Y. Wang, "Structadmm: Achieving ultrahigh efficiency in structured pruning for dnns," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 2259–2273, 2022.

[38] M. Lin, R. Ji, S. Li, Y. Wang, Y. Wu, F. Huang, and Q. Ye, "Network pruning using adaptive exemplar filters," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 7357–7366, 2022.

[39] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *7th International Conference on Learning Representations*, 2019.

[40] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal sgd for pruning very deep convolutional networks with complicated structure," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4938–4948.

[41] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations*, 2015.

[42] PyTorch, *Torchvision Official Models*, 2020. [Online]. Available: https://pytorch.org/docs/stable/torchvision/models.html

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

[44] 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)*, vol. 28, no. 1, pp. 100–108, 1979.

[45] Y. 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," in *4th International Conference on Learning Representations*, 2016.

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

[47] J. H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *IEEE International Conference on Computer Vision*, 2017, pp. 5068–5076.

[48] C. Wang, R. Grosse, S. Fidler, and G. Zhang, "Eigendamage: Structured pruning in the kronecker-factored eigenbasis," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6566–6575.

[49] C. Jiang, G. Li, C. Qian, and K. Tang, "Efficient dnn neuron pruning by minimizing layer-wise nonlinear reconstruction error," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. AAAI Press, 2018, pp. 2298–2304.

[50] X. Ding, G. Ding, Y. Guo, J. Han, and C. Yan, "Approximated oracle filter pruning for destructive cnn width optimization," in *International Conference on Machine Learning*. PMLR, 2019, pp. 1607–1616.

[51] Q. Huang, K. Zhou, S. You, and U. Neumann, "Learning to prune filters in convolutional neural networks," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 709–718.

[52] P. Singh, V. S. R. Kadi, N. Verma, and V. P. Namboodiri, "Stability based filter pruning for accelerating deep cnns," in *IEEE Winter Conference on Applications of Computer Vision*, 2019, pp. 1166–1174.

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

[54] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 4340–4349. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/He_Filter_Pruning_via_Geometric_Median_for_Deep_Convolutional_Neural_Networks_CVPR_2019_paper.html

[55] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. S. Doermann, "Towards optimal structured CNN pruning via generative adversarial learning," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 2790–2799. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Lin_Towards_Optimal_Structured_CNN_Pruning_via_Generative_Adversarial_Learning_CVPR_2019_paper.html

[56] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," *CoRR*, vol. abs/2002.10179, 2020. [Online]. Available: https://arxiv.org/abs/2002.10179

[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] X. Xu, M. S. Park, and C. Brick, "Hybrid pruning: Thinner sparse networks for fast inference on edge devices," *arXiv preprint arXiv:1811.00482*, 2018.

[59] D. Wang, L. Zhou, X. Zhang, X. Bai, and J. Zhou, "Exploring linear relationship in feature map subspace for convnets compression," *arXiv preprint arXiv:1803.05729*, 2018.

[60] J. Luo, H. Zhang, H. Zhou, C. Xie, J. Wu, and W. Lin, "Thinet: Pruning CNN filters for a thinner net," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 10, pp. 2525–2538, 2019. [Online]. Available: https://doi.org/10.1109/TPAMI.2018.2858232

[61] PyTorch, *PyTorch Official Example*, 2020. [Online]. Available: https://github.com/pytorch/examples/blob/master/imagenet/main.py

[62] B. Hariharan, P. Arbelaez, L. D. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, D. N. Metaxas, L. Quan, A. Sanfeliu, and L. V. Gool, Eds. IEEE Computer Society, 2011, pp. 991–998. [Online]. Available: https://doi.org/10.1109/ICCV.2011.6126343

[63] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, 2018. [Online]. Available: https://doi.org/10.1109/TPAMI.2017.2699184

[64] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 6230–6239. [Online]. Available: https://doi.org/10.1109/CVPR.2017.660

[65] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[66] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.