Accelerating Convolutional Neural Networks by Group-wise 2D-filter Pruning

Niange Yu
Department of Computer
Sicence and Technology
Tsinghua University
Beijing 100084, China
yng15@mails.tsinghua.edu.cn

Shi Qiu
SenseTime Group Limited
Shatin, Hong Kong
sqiu@sensetime.com

Xiaolin Hu, Jianmin Li
Department of Computer
Sicence and Technology
Tsinghua National Laboratory
for Information Science and Technology
Tsinghua University
Beijing 100084, China
{xlhu,lijianmin}@tsinghua.edu.cn

***Abstract*—Network pruning is an effective way to accelerate Convolutional Neural Networks (CNNs). In recent years, structured pruning methods are proposed in favor of unstructured methods as they have shown greater speedup in practical use. Existing structured methods does pruning along two main dimensions: 3D-filter wise, i.e., remove a 3D-filter as a whole, and filter-shape wise, i.e., remove a same position from all 3D-filters. In this work, we propose a new group-wise 2D-filter pruning approach that is orthogonal and complementary to the existing methods. The proposed approach removes a portion of 2D-filters from each 3D-filter according to the pruning patterns learned from the data, and leads to compressed models that do not require sophisticated implementation of convolution operations. A fine-tuning process is followed to recover the accuracy. The knowledge distillation (KD) framework is explored in the fine-tuning process to improve the performance. We present our method for learning the pruning pattens as well as the fine-tuning strategy based on knowledge distillation. The proposed approach is validated on two representative CNN models – ZF and VGG16, pre-trained on ILSVRC12. Experimental results demonstrate the effectiveness of our approach. In VGG16, we get even higher accuracy after speeding-up the network by 4 times.**

I. Introduction

Convolutional Neural Networks (CNNs) have achieved state-of-the-arts performance in a wide spectrum of computer vision tasks, including object recognition [1][2][3][4], object detection [5][6], object segmentation [7][8] and so on. Recent researches on the design of CNN architectures advocated the use of deep convolutional architectures that were often comprised of tens to hundreds of convolutional layers. While such a design choice continued to boost the CNNs' accuracy, it also dramatically increased the computation cost, which hindered the deployment of CNN-based applications, especially on embedded and mobile devices where computation power was limited.

Recently, considerable research efforts have been devoted to accelerate CNNs, including low rank approximation [9][10][11][12], network pruning [13][14][15], sparsity regularization [16][17][18], parameter quantization [19][20] and network binarization [21][22]. Parameter quantization and network binarization methods took advantages of low bit width representation on weight, activation and gradient of the network to accelerate it and reduce the memory burden. These approaches' performance on acceleration were highly dependent on how much bit width they used and the hardware platform they ran on. In low rank approximation method, the weight tensors in the convolutional layers were decomposed and approximated by a product of smaller factors. It could gain practical speedups with commonly used convolution implementations regardless of the hardware platforms.

Network pruning and sparsity regularization accelerate CNNs by removing some connections from the models. Han et al. proposed an approach that iterates between pruning the weights and fine-tuning the pruned model, and obtained a compressed AlexNet [1] with over 90% of the weights pruned and no loss in accuracy [13][23]. However, since filters after prunning were irregular and unstructured, one need to implement sophisticated convolution operation to get actual acceleration from the pruned model. To address this issue, Lebedev & Lempitsky [17] proposed to add a group-sparsity regularizer when conducting pruning. The pruned 3D-filters shared the same shape though the new shape was irregular. This approach could be viewed as changing the shape that all 3D-filter shared, so it was called shape-wise pruning. Apart from adding some constrains, another approach to tackle it was to use structured pruning methods like 3D-filter wise or layer wise pruning [14][18][15]. Rather than single weight, the pruning candidates of these methods were the whole 3D-filter or the whole layer respectively. While layer wise pruning could only be applied in networks with a "short-cut" design like ResNet [4], 3D-filter wise pruning was a more general method.

In previous works on 3D-filter wise pruning, the pruning ratio was often limited, because removing 3D-filters would consequently remove the corresponding feature-maps and drastically reduce the layer's representation power. In this paper, we view 3D-filters as a set of 2D-filters, and propose a group-wise 2D-filter pruning approach to compress 3D-filters. More specifically, as shown in Fig 1, the 3D-filters are equally divided into g ($g > 1$) groups ($g = 2$ in Fig 1). All 3D-filters within a same group are forced to have the same pruning pattern (i.e., the indices of the channels

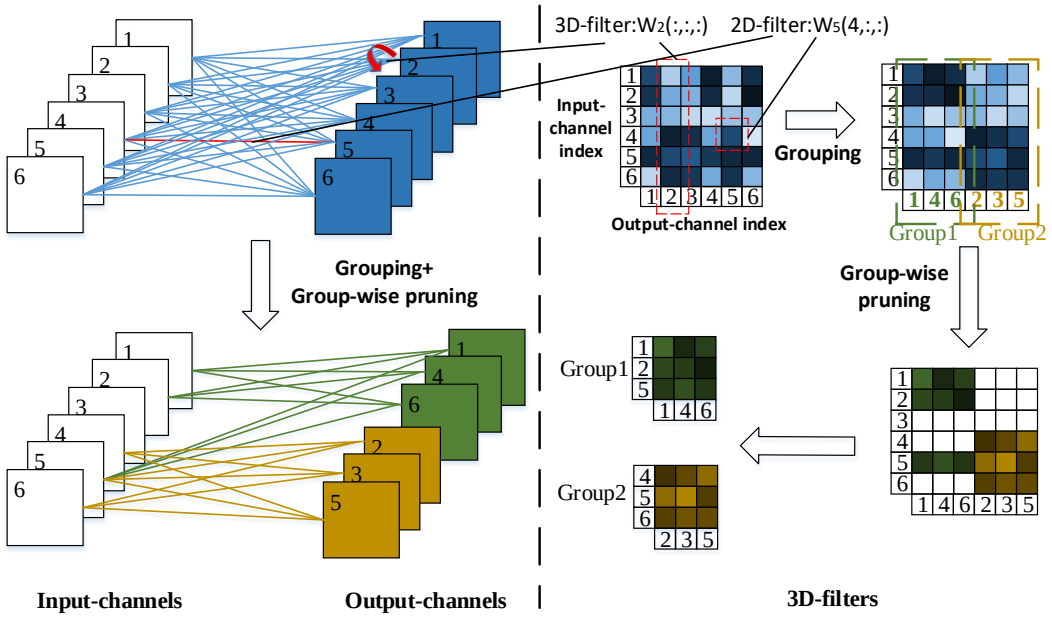


Fig. 1: Group-wise pruning. We group the 3D-filters into two and prune each group's input-channel by a ratio of 50%. The network connection between the input and output channels before and after pruning is shown on the left. On the right, we show the transform of the 3D-filters along the process. At the beginning, all the original 3D-filters are organized in a $T \times S$ grid (In this case, $T = 6, S = 6$), each entry of which represents 2D-filter $W_t(s, :, :)$, and each column represents a 3D-filter consisting of S 2D-filters. The darkness of the entries indicates the corresponding 2D-filter's importance in the network. After grouping, the columns in the grid are shuffled as indicated by the index at the bottom. In this case, the 3D-filters are divided into two groups, the first group consists of the 1-th, 4-th and 6-th 3D-filters and the rest are in the second group. After that, the input channels in each group are pruned by a ratio of 50% according to its importance of the group (i.e., the corresponding 2D-filters' importance). The remaining 2D-filters in each groups can then be re-orgnized into a new compact grid.

to be pruned in each 3D-filter). Pruning patterns of different groups can be different. The pruning pattern between the input-channels and output-channels are illustrated in Fig 1, also shown is the shape of the 3D-filters along the pruning process. By adjusting a shared pruning ratio p among all groups ($p = 0.5$ in Fig 1), we are able to control the computation cost of the resulting model. Our proposed approach enjoys a number of advantages: (1) Since we prune 2D-filters instead of 3D-filters, the finer pruning granularity leads to higher sparsity (pruning ratio). (2) Our approach is able to retain the number of feature-maps in each layer, and therefore better approximate the output of original model. (3) The group-wise pruning leads to highly structured filters, which is easy to implement and bring practical acceleration. Since the large convolution kernel in the original network is reduced to g small kernels by our methods, convolution kernel in each group perform convolution with a subset of the input-channels. The convolution operation implemented within each group remains the same as the original network.

Identifying groups of 3D-filters and the pruning pattern for each group is crucial to our approach. To achieve this, we first evaluate each 2D-filter's importance with respect to pruning. Inspired by the work in [24], which evaluated 3D-filter's importance based on the feature map it produced, we proposed to evaluate 2D-filter's importance based on the intermediate feature map it produced. The intermediate feature map's sensitivity to the input as well as its impact on the network's final output are both taken into consideration. After we obtain each single 2D-filter's importance, all the 3D-filters are clustered into several groups such that 3D-filters from the same group share indices of their top important 2D-filters (see Fig 1). Then, we can identify the least important indices within each 3D-filter group, and remove the corresponding 2D-fitlers. This process of identifying groups of 3D-filters and finding the pruning pattern for each group is optimized in an alternative way as illustrated in Algorithm1. After the groups and the pruning patterns are determined, the actual pruning is only to be carried out once.

To further improve the accuracy, we explore the Knowledge Distillation (KD) [25] for fine-tuning the pruned model. We show empirically that compared with the traditional fine-tuning methods, KD could bring a consistent significant improvement.

In summary, our contributions are three-fold: (1) We propose to accelerate the computation of convolutional layers by group-wise pruning the 2D-filters in each 3D-filter. The pruned 3D-filters are highly structured and easy to be translated to actual speedups. (2) We design a method for estimating the importance of 2D-filters, based on which 3D-filters are grouped and pruned. (3) We present a practical and effective strategy for fine-tuning the pruned networks, which further imporves the accuracy of the pruned model.

II. METHOD

Our group-wise 2D-filter pruning approach consists of three key steps. (1) 2D-filter evaluation, where 2D-filter's importance are evaluated in the context of pruning; (2) Group-wise pruning, where the 3D-filters are clustered into groups and each group's pruning pattern is determined; (3) Fine-tuning with knowledge distillation (KD), where the pruned CNN model is further fine-tuned with KD.

A. 2D-filter evaluation

The weights of a convolutional layer consists of a bunch of 3D-filters $W_t \in R^{C\times N\times M}(1 \le t \le D)$ where D is the number of 3D-filter and C, N, M denote the spatial dimensions of the 3D-filter. Each 3D-filter perform convolution operation with the input feature-maps which can be viewed as a 3D-tensor $Z \in R^{C\times U\times V}$ and produce an output feature-map $A_t \in R^{U\times V}(1 \le t \le D)$:

$$\begin{aligned} A_t(x,y) = &\sum_{s=1}^{C}\sum_{i=1}^{N}\sum_{j=1}^{M}[W_t(s,i,j)\cdot \\ &Z(s, x+i-\frac{N+1}{2}, y+j-\frac{M+1}{2})] + b_t \end{aligned} \quad . \quad (1)$$

To simplify the formulation, we assumed the input and output feature map have the same size $U \times V$. In the equation above, $1 \le x \le U$, $1 \le y \le V$, b_t is the bias. While a 3D-filter $W_t \in R^{C\times N\times M}$ consists of a set of 2D-filters $W_t(s,:,:) \in R^{N\times M}(1 \le s \le C)$, the 3D convolution as in Eqn (1) can be viewed as a composition of 2D convolutions:

$$A_t(x,y) = \sum_{s=1}^{C} H_{ts}(x,y) + b_t, \quad (2)$$

$$\begin{aligned} H_{ts}(x,y) = &\sum_{i=1}^{N}\sum_{j=1}^{M}[W_t(s,i,j)\cdot \\ &Z(s, x+i-\frac{N+1}{2}, y+j-\frac{M+1}{2})], \end{aligned} \quad (3)$$

where $H_{ts} \in R^{U\times V}$ is the 2D convolution output produced by $W_t(s,:,:)$. We evaluate the importance of 2D-filter $W_t(s,:,:)$ based on H_{ts}. H_{ts} is called "intermediate feature-map" in the following contents.

In fact, intermediate feature-map H_{ts}'s value depends on the input of the network. We denote H_{ts} as $H_{ts}(X_k)$ given the input X_k, thus, $H_{ts}(X_k)$ changes for different X_k. $H_{ts}(X_k)$'s sensitivity to the changes of input X_k can be viewed as its discrimination ability to different inputs. We use the variance of the set $H_{ts}(X_k), k = (1, 2, \ldots, K)$ (K is the input case number in our measurement, each element inside $H_{ts}(X_k)$ is viewed as a variable) to measure the discrimination ability of $H_{ts}(X_k)$ and denote it as E_{ts}.

$$E_{ts} = \frac{\sum_{k=1}^{K}\sum_{u=1}^{U}\sum_{v=1}^{V}(H_{ts}(X_k)[u,v] - a_{ts})^2}{K\times U\times V}, \quad (4)$$

$$a_{ts} = \frac{\sum_{k=1}^{K}\sum_{u=1}^{U}\sum_{v=1}^{V}H_{ts}(X_k)[u,v]}{K\times U\times V}, \quad (5)$$

where a_{ts} is the mean value of set $H_{ts}(X_k), k = (1, 2, \ldots, K)$. On the other hand, we want the pruning of the intermediate feature map H_{ts} bring minimal impact to the output of network as we want to maintain the original accuracy. A good way to measure the sensitivity to H_{ts} of network is the magnitude of the gradient of loss function with respective to H_{ts}. When we evaluate the importance of H_{ts}, we want to take both the sensitive of H_{ts} with respect to the inputs and the output's sensitivity with respect to H_{ts} into account. Thus, we modify the Eqn (4) as:

$$E_{ts} = \frac{\sum_{k=1}^{K}\sum_{u=1}^{U}\sum_{v=1}^{V}|f'_{(t,s,u,v)}(H(X_k))|(H_{ts}(X_k)[u,v] - a_{ts})^2}{\sum_{k=1}^{K}\sum_{u=1}^{U}\sum_{v=1}^{V}|f'_{(t,s,u,v)}(H(X_k))|}, \quad (6)$$

where $f(H(X_k))$ denotes the network loss function, $f'_{(t,s,u,v)}(H(X_k))$ is the gradient of $f(H(X_k))$ with respect to $H_{ts}(X_k)[u,v]$. After that, we attempt to further use $|f'_{(t,s,u,v)}(H(X_k))|$ to weight the computing of a_{ts} as :

$$a_{ts} = \frac{\sum_{k=1}^{K}\sum_{u=1}^{U}\sum_{v=1}^{V}|f'_{(t,s,u,v)}(H(X_k))|H_{ts}(X_k)[u,v]}{\sum_{k=1}^{K}\sum_{u=1}^{U}\sum_{v=1}^{V}|f'_{(t,s,u,v)}(H(X_k))|}. \quad (7)$$

The experiments show that with the new formulation, it can get better performance.

While pruning filter $W_t(s,:,:)$(i.e., set $W_t(s,:,:) = 0$) directly would have intermediate feature-map H_{ts} result in zero for any input X_k. As an example, if filter $W_t(p,:,:)$ was pruned, the corresponding output feature-map given input X_k would result in

$$\begin{aligned} A_t^{(p)}(X_k) &= \sum_{s\neq p} H_{ts}(X_k) + H_{tp}(X_k) + b_t \\ &= \sum_{s\neq p} H_{ts}(X_k) + 0 + b_t. \end{aligned}$$

Thus, the network's output would change a lot for input X_k because most values in $H_{tp}(X_k)$ are far from 0. When we compute E_{ts}, a_{ts} is produced as a by-product. While a_{ts} is the weighted mean value for H_{ts}, it would be beneficial to replace 0 by a_{ts}. After that, the new output feature-map is

$$A_t^{(p)}(X_k) = \sum_{s\neq p} H_{ts}(X_k) + a_{tp} + b_t = \sum_{s\neq p} H_{ts}(X_k) + \tilde{b_t}.$$

Thus, $\tilde{b_t}$ is the new bias for the 3D-filter. In fact, each time we prune a 2D-filter $W_t(s,:,:)$, a_{ts} would be added to the corresponding bias b_t.

B. Group-wise pruning

When 2D-filters are pruned without any constrains, we end up with a set of 3D-filters with different shapes. However, most efficient convolution operation implementations require that all 3D-filters share the same shape.

To tackle this issue, we divide the 3D-filters into g groups, the filters in the same group are required to be pruned at the same channels, while 3D-filters from different groups are not subject to this constrains. The process can be simply expressed as group the 3D-filters first, and then prune the 2D-filters in each group according to the selected input-channel. The process is shown at Fig 1. Each 2D-filter is identified by (t, s), our objective function is :

$$\begin{aligned} &SE = \sum_{t}\sum_{s} E(t,s), (t,s) \in P \\ &s.t. |P| = D \times C \times p. \end{aligned} \tag{8}$$

In Eqn (8), $E(t, s)$ equal to E_{ts} in Eqn (6). P denotes the set consists of all the 2D-filter pruned. $D \times C$ is the total 2D-filter number, p denotes the pruning ratio we adopt.

While our goal is to find P that minimize SE and satisfy the group constrain, an alternative optimization algorithm is proposed for that purpose. In brief, we first give an initial partition of 3D-filters by K-means, then each group is given a pruning pattern that minimize SE under this partition. While this is not the optimal solution because the partition of 3D-filters is not optimized, we re-divided the 3D-filters into groups based on the pruning pattern in each group which is the optimal solution under the previous 3D-filter partition. After that, we would give a new pruning pattern under the new 3D-filter partition again. This is a alternative optimization process between dividing 3D-filters into groups and giving each group's pruning pattern. It will iterate for many times until SE converged. The details are shown in Algorithm 1.

C. Fine-tuning with knowledge distillation

To improve the fine-tuning process, we explore the knowledge distillation framework. We denote student model as S and the teacher as T, Their softmax output can be formulated as $P_S = softmax(A_S(X, W_S))$ and $P_T = softmax(A_T(X, W_T))$ respectively. $A_S(X, W_S)$ and $A_T(X, W_T)$ are the inputs to the softmax layer in the network. To make the output of the softmax contain more information, KD introduce a soft version of softmax transform as :

$$\tilde{P_S} = softmax(\frac{A_S(X, W_S)}{t}), \tag{9}$$

$$\tilde{P_T} = softmax(\frac{A_T(X, W_T)}{t}). \tag{10}$$

A new parameter t is introduced to control the degree of "soften" about the output. Then, the student network will be trained with the soft teacher output together with the true label denoted as y. The loss function of the framework can be formulate as :

$$L_{KD} = H(y, P_S) + \lambda H(\tilde{P_T}, \tilde{P_S}). \tag{11}$$

Algorithm 1 group-wise pruning

Input:

$E \in R^{D \times C}$: $E(t, s)$ denotes the importance of 2D-filter $W_t(s, :, :)$;

p: pruning ratio;

g: group number.

Output:

P: the set consists of 2D-filters pruned.

1: Initialize $I_w^{(0)}$ and $J_w^{(0)} (w \in \{1, 2, \ldots, g\})$. Cluster vector set $\{E(1, :), E(2, :), \ldots, E(D, :)\}$ into g groups by K-means cluster algorithm, each vector refer to a 3D-filter, thus we get an initial 3D-filter division. $I_w^{(0)}$ denotes the set consists of index of all 3D-filters in group w;
Then, we select the top $C \times p$ most unimportant input-channel set $J_w^{(0)}$ for each group. The importance of s-th input-channel in group w is $\sum_{t \in I_w^{(0)}} E(t, s)$

$$\{I_1^{(0)}, I_2^{(0)}, \ldots, I_g^{(0)}\} \leftarrow \{1, 2, \ldots, D\}$$

$$J_w^{(0)} = subset(\{1, 2, \ldots, C\}), w \in \{1, 2, \ldots, g\}$$

2: **Repeat:**

3: Update the prune pattern in each group, i.e., re-select the most unimportant input-channels for each group.

$$\{J_1^{(z)}, J_2^{(z)}, \ldots, J_g^{(z)}\} \leftarrow \{J_1^{(z-1)}, J_2^{(z-1)}, \ldots, J_g^{(z-1)}\}$$

4: Reassign the 3D-filters in groups(i.e., update I_w^{z-1} to I_w^z) based on $J_w^{(z)}$:

$$\{I_1^{(z)}, I_2^{(z)}, \ldots, I_g^{(z)}\} \leftarrow \{I_1^{(z-1)}, I_2^{(z-1)}, \ldots, I_g^{(z-1)}\}$$
$$t \in I_w^{(z)}, w = \underset{w'}{\operatorname{argmin}} \sum_{s \in J_{w'}^{(z)}} E(t, s)$$

5: Compute the total pruning loss:

$$SE^{(z)} = \sum_{w=1}^{g} \sum_{t \in I_w^{(z)}} \sum_{s \in J_w^{(z)}} E(t, s)$$

6: **Until** $SE^{(z)} = SE^{(z-1)}$

7: **Return** $P = \{W_t(s, :, :) | t \in I_w^{(z)}, s \in J_w^{(z)}, w = (1, 2, \ldots, g)\}$

H is the cross-entropy function and λ is a parameter introduced to balance the two items. In this paper, we call the first item "hard-loss" and the second "soft-loss".

A problem is whether to use dropout in fully-connected layers. Dropout plays an indispensable role in reducing over-fitting in big CNN models, but it will result in unstable output of the network. Since we need to learn from the teacher, its output must be consistent in different training stage. Thus, dropout should be removed from teacher. As for the student, our experiments showed that dropout would hurt the

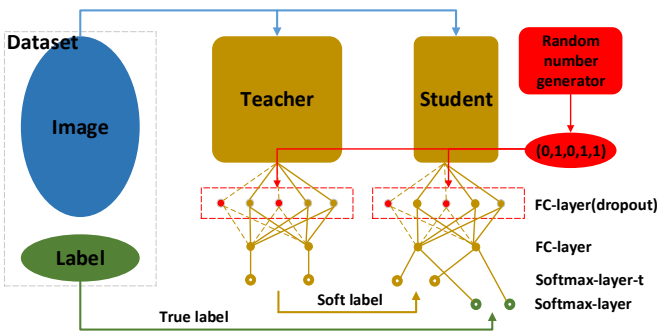


Fig. 2: The process of fine-tuning with knowledge distillation

performance if only the soft-loss is applied in training. But if we removed dropout when hard-loss was applied, the network tended to be over-fitting. It seems that there is a conflict whether to use dropout between soft-loss and hard-loss. When we used both loss together as KD suggests, no matter whether the dropout was used, the performance was lower comparing to the case using soft-loss alone without dropout.

One explanation about dropout is that it learns an ensemble because at every iteration only a sub-network is updated. The efficiency of using an ensemble of models partly comes from the diversity among the models. Compared with one-hot label, using soft label to optimize the student reduced the diversity of the ensemble in student as the soft label contains richer information and is consistent when facing different sub-networks in the ensemble, so that the diversity of ensemble could potentially reduced. To overcome this problem, we propose to introduce dropout in teacher so as to optimize sub-networks of the student with the soft label generated by sub-networks of the teacher. The teacher and the student share the same sub-network structure by keeping the dropout nuerons be the same for them at every iteration. By doing so, sub-networks of student would have the same diversity as teacher. An illustration of this step is shown in Fig 2.

III. Experiments

All our experiments were conducted on the ILSVRC12 dataset which consists of over one million images in training and validation set [26]. Because Alex-net [1] has "group feature" [1] in some convolutional layers on its own, we chose ZF-net [27] as our basic CNN model instead to ease our exploration. In addition, we further verified our method on a deeper and more widely used VGG16 network.

A. Pruning

The architecture of ZF-net is similar to Alex-net except for two points, one is removing the group feature in all layers, the other is replacing the 11×11 kernel with 7×7 kernel in the first convolutional layer. The baseline ZF-net was trained from scratch by ourselves with a top-1 accuracy of 60.87%. Besides, the VGG16 was directly downloaded from Caffe's model zoo with a top-1 accuracy of 68.36%.

We performed group-wise 2D-filter pruning on all convolutional layers in the networks except for the first one. First, we used our method to evaluate all the 2D-filter's importance based on 50,000 images sampled from ILSVRC12 training set (i.e., set $K = 50,000$). The evaluation was based on Eqn (6) and Eqn (7) could be obtained by a single forward and backward propagation for each image. We then simply remove the least important 2D-filters. We compared our method with the magnitude based methods. Similar to the 3D-filter evaluation method proposed by Li et al. [15], we used the sum of all magnitude of the 2D-filter's weights as their importance. We pruned the network based on the two methods at varying pruning ratios and compared the accuracy of the pruned networks. The results were shown in Fig 3 that our method surpassed the magnitude base method significantly.

Based on the evaluation, we pruned the 2D-filters in each 3D-filter at a ratio of 0.5 according to their importance without any constrains, which produced an unstructured sparse network denoted as "ZF-pruned_0". While it could not get practical acceleration, we used it to explore KD in the early experiments. For group-wise 2D-filter pruning, we used Algorithm 1 to divide the 3D-filters into g groups and pruned the 2D-filters in each group by a ratio of p. Compared with pruning without group constrain like "ZF-pruned_0", it would result in a bigger pruning loss SE. When we set $p = 0.5$ and $g = 2$ for group-wise pruning on ZF-net, over the iterations of Algorithm 1, a relative magnitude of SE at each iteration with respect to the SE of "ZF-pruned_0" (i.e., $p = 0.5$ and no group constrain) was shown at Fig 4, which illustrated that our algorithm was convergent. We experimented on three configuration by the combination of p and g for ZF-net and VGG16 respectively. The corresponding pruned networks' architecture were shown at Table I. Note that for each input-channels, if it was pruned in all groups, it became void in the network and the corresponding 3D-filter in the previous layer could be removed. This resulted in the changes in the number of 3D-filter number in some layers.

B. Fine-tuning

We set the standard training process proposed in Alex-net [1] as our baseline fine-tuning method. The learning rate was set to $\eta = 10^{-4}$. Over the training procedure, it decreased by a factor of 10 every 10 epochs. Some of the pruned networks in Table I were fine-tuned by the baseline method, results are listed in Table II marked as "(hardloss only)".

A key problem regarding knowledge distillation was how to get soft label from teacher. A naive method was storing all soft label getting from teacher as true label for student. The disadvantage was that we could not use data augmentation such as random crop on input image because it would make the input of student inconsistent with that of teacher. The other method was that the same augmented data was fed to both teacher and student, then the soft label was generated online by teacher and used as label for student. We compared the

[1]The 2,4,5-th convolutional layers in Alex-net have their 3D-filters divided into two groups, each group perform convolution with half of the input channels. It's a special case of our approach.

TABLE I: Pruned network architecture for ZF-net and VGG16. All the layer entries are specified in a S-T format (S: 2D-filter number in each 3D-filter, T: 3D-filter number in this layer). Sparsity is computed only for the convolutional layers.

models (p/g)	conv1	conv2	conv3	conv4	conv5	sparsity
ZF-net	3-96	96-256	256-384	384-384	384-256	1.0
ZF-pruned_0	3-96	48-256	128-384	192-384	192-256	0.50
ZF-pruned-group-wise_1 **(0.5/2)**	3-76	48-184	128-265	192-326	192-256	0.41
ZF-pruned-group-wise_2 **(0.5/1)**	3-48	48-128	128-192	192-192	192-256	0.31
ZF-pruned-group-wise_3 **(0.75/4)**	3-60	24-168	64-232	96-300	96-256	0.19
VGG16	3-64 64-64	64-128 128-128	128-256 256-256 256-256	256-512 512-512 512-512	512-512 512-512 512-512	1.0
vgg16-pruned-group-wise_0 **(0.5/1)**	3-32 32-32	32-64 64-64	64-128 128-128 128-128	128-256 256-256 256-256	256-256 256-256 256-512	0.25
vgg16-pruned-group-wise_1 **(0.6/1)**	3-26 26-26	26-52 52-52	52-103 103-103 103-103	103-205 205-205 205-205	205-205 205-205 205-512	0.20
vgg16-pruned-group-wise_2 **(0.7/1)**	3-20 20-20	20-39 39-39	39-77 77-77 77-77	77-154 154-154 154-154	154-154 154-154 154-512	0.12

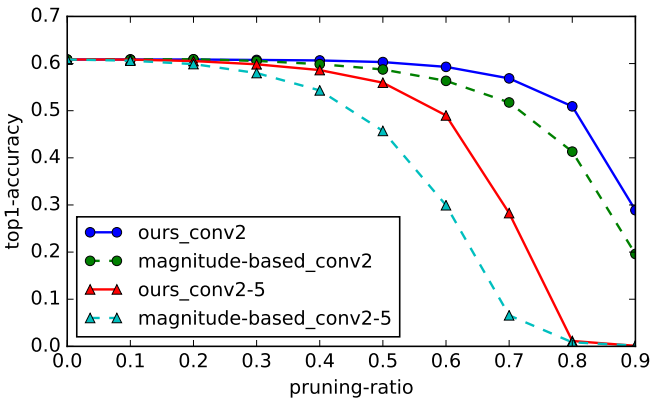


Fig. 3: Performance w.r.t the pruning ratio. *conv2* indicates pruning the second convolutional layer only. *conv2-5* indicates pruning all the four layers at the same ratio.

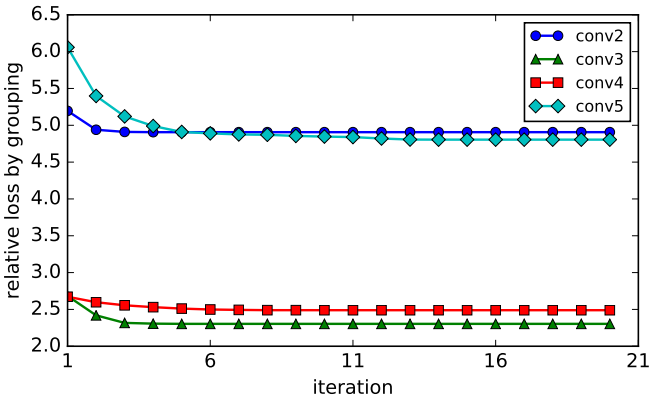


Fig. 4: Relative pruning loss with grouping constraint in Algorithm1. Relative pruning loss is the total pruning loss with group constraint over the total pruning loss without group constraint.

result of these two methods as shown in Fig 5a and adopted the second method because of its higher performance.

In previous explorations, we found the best setting for "temperature" t in Eqn (9) and Eqn (10) was 2. The gradients produced by soft-loss had lower magnitude than the hard-loss which enabled a higher learning rate $\eta = 10^{-3}$. After that, we used both the soft-loss and hard-loss at fine-tuning as KD, the parameter λ that control the relevant influence between the two items was set to 10. Then we found the issue about "dropout" described in section II-C, the training process of three different configuration of dropout, i.e., without dropout, with dropout only in student and our new dropout method were shown at Fig 2. Our new dropout approach had the leading performance.

The fine-tuning results of all the pruned networks in Table I by KD were summarized at Table II (without any mark). Fine-tuning with KD had significantly better performance than traditional approach for all pruned networks. Besides, we attempted to train ZF-pruned-group-wise_2 from scratch with the same training procedure of original ZF-net (marked as "(training from scratch)" at Table II). It was surprising to find that its accuracy is higher than fine-tuning ZF-pruned-group-wise_2 in the traditional way, but still lower than our method. We also tried to use KD to train it from scratch using the same configuration (marked as "(KD from scratch)") but got even lower accuracy comparing with "(training from scratch)".

Compared to 3D-filter pruning method proposed by Li et al. [15], we got significantly better accuracy (-1.9% compared to -3.2%) under the same speed-up ratio (3.13×). A comparison with the latest low rank approximation methods on VGG16 were also conducted and listed at Table II. vgg16-pruned-group-wise_0 and vgg16-pruned-group-wise_1 had comparable performances compared with [11] and [12] respectively. For vgg16-pruned-group-wise_0, we got higher accuracy than the original VGG16, it indicated that after pruning the unimportant filters, the network got a better generalization. Besides, vgg16-pruned-group-wise_2 exhibited an amazing 10.04× speed-up with a comparable accuracy

TABLE II: Fine-tuning result of different networks. The weight-reduction and speed-up are computed concerning only the convolutional layers.

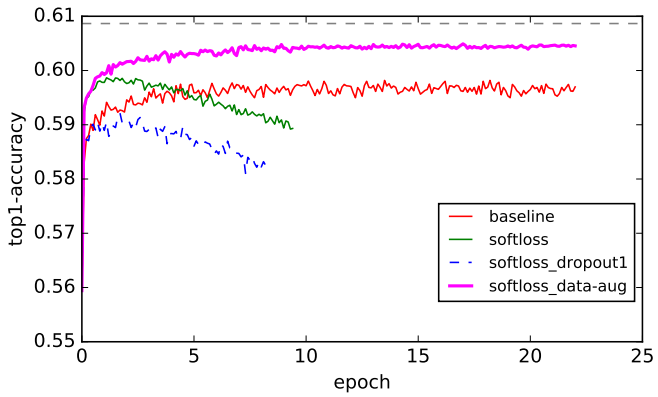
methods	weight-reduction	speed-up	accuracy
ZF			
ZF-pruned_0	1.99	1.0	**-0.3%**
ZF-pruned_0 (hardloss only)	1.99	1.0	-1.1%
ZF-pruned-group-wise_1	2.41	2.24	**-1.1%**
ZF-pruned-group-wise_2	3.21	3.13	**-1.9%**
ZF-pruned-group-wise_2 (hardloss only)	3.21	3.13	-3.0%
ZF-pruned-group-wise_2 (training from scratch)	3.21	3.13	-2.6%
ZF-pruned-group-wise_2 (KD from scratch)	3.21	3.13	-3.4%
ZF-pruned-group-wise_3	5.15	4.17	**-3.1%**
Li et al.[15]: 3D-filter pruning	3.21	3.13	-3.2%
VGG16			
Tai et al.[11]:low rank approximation	2.75	3.10	-0.29%
Kim et al.[12]:low rank approximation	5.22	5.03	-0.5%
vgg16-pruned-group-wise_0	4.0	3.98	**+0.46%**
vgg16-pruned-group-wise_0 (hardloss only)	4.0	3.98	-0.06%
vgg16-pruned-group-wise_1	5.02	5.85	**-1.0%**
vgg16-pruned-group-wise_1 (hardloss only)	5.02	5.85	-2.1%
vgg16-pruned-group-wise_2	8.04	10.04	**-2.9%**
vgg16-pruned-group-wise_2 (hardloss only)	8.04	10.04	-6.1%

degradation, similar result had never been reported before.

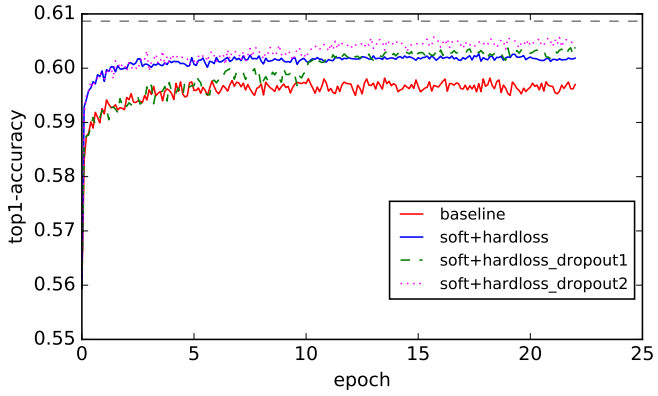
In group-wise pruning process, if we set $g = 1$, our method is equal to 3D-filter pruning. We find that in our approach, under the same pruning ratio, pruning 3D-filter directly would result in a larger accuracy drop but a higher speed-up ratio (ZF-pruned-group-wise_2: $-1.9\%/3.13\times$ compared to ZF-pruned-group-wise_1: $-1.1\%/2.24\times$). For ZF-pruned-group-wise_2, it has only half the channel in each layer of ZF-net which greatly reduces the layer's representation power. But owning to the efficiency of knowledge distillation in the fine-tuning stage, we can restore its accuracy to the degree that we can tolerate. Whether to use $g = 1$ is up to a trade off between the accuracy and speed-up. Note that for the experiments on VGG16, we use only $g = 1$. The reason comes in two parts, one is that the $g = 1$ cases were enough to exhibit the efficiency of our approach compared to other methods, and the other is that it takes more time to fine-tune the pruned networks obtaining from $g > 1$ cases because of our implementation. We leave the case of $g > 1$ in VGG16 or even larger networks for future works.

IV. Conclusion

In this paper, we propose a new structured network pruning approach to accelerate convolutional neural network and explore the knowledge distillation framework in fine-tuning the pruned network. We introduce a new 2D-filter evaluation method to locate the pruning candidates in a data driven way and this method outperform the magnitude based method significantly. The experiments show that our approach has a competitive performance compared with previous approaches on accelerating large CNN models.



(a) Fine-tuning with soft-loss. "softloss" means fine-tuning without dropout or data augmentation. "softloss_dropout1" means with dropout only and "softloss_data-aug" means the soft label is produced online with random crop and mirror.



(b) Fine-tuning with both soft-loss and hard-loss. "soft+hardloss" means fine-tuning without dropout. "dropout1" means traditional dropout and "dropout2" indicates the proposed dropout method illustrate in Fig 2.

Fig. 5: Loss in the fine-tuning process. "baseline" traditional fine-tuning approach without soft label.

REFERENCES

[1] 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.

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

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[7] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[8] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr, "Conditional random fields as recurrent neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1529–1537.

[9] 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.

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

[11] C. Tai, T. Xiao, X. Wang *et al.*, "Convolutional neural networks with low-rank regularization," *arXiv preprint arXiv:1511.06067*, 2015.

[12] 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.

[13] 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.

[14] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *arXiv preprint arXiv:1512.08571*, 2015.

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

[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] V. Lebedev and V. Lempitsky, "Fast convnets using group-wise brain damage," *arXiv preprint arXiv:1506.02515*, 2015.

[18] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *arXiv preprint arXiv:1608.03665*, 2016.

[19] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2014, pp. 1–6.

[20] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1131–1135.

[21] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.

[22] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *arXiv preprint arXiv:1603.05279*, 2016.

[23] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR, abs/1510.00149*, vol. 2, 2015.

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

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

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[27] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision*. Springer, 2014, pp. 818–833.