Leveraging Filter Correlations for Deep Model Compression

Pravendra Singh* Vinay Kumar Verma* Piyush Rai Vinay P. Namboodiri
Department of Computer Science and Engineering, IIT Kanpur, India
{psingh, vkverma, rpiyush, vinaypn}@iitk.ac.in

Abstract

We present a filter correlation based model compression approach for deep convolutional neural networks. Our approach iteratively identifies pairs of filters with the largest pairwise correlations and drops one of the filters from each such pair. However, instead of discarding one of the filters from each such pair naïvely, the model is re-optimized to make the filters in these pairs maximally correlated, so that discarding one of the filters from the pair results in minimal information loss. Moreover, after discarding the filters in each round, we further finetune the model to recover from the potential small loss incurred by the compression. We evaluate our proposed approach using a comprehensive set of experiments and ablation studies. Our compression method yields state-of-the-art FLOPs compression rates on various benchmarks, such as LeNet-5, VGG-16, and ResNet-50,56, while still achieving excellent predictive performance for tasks such as object detection on benchmark datasets.

1. Introduction

Recent advances in convolutional neural networks (CNN) have yielded state-of-the-art results on many computer vision tasks, such as classification, detection, etc. However, training data and storage/computational power requirements limit their usage in many settings. To address this, one line of research in this direction has focused on training CNNs with limited data [10, 57, 21, 42, 40, 60]. Another line of work has focused on model compression to make it more efficient in terms of FLOPs (speedup) and memory requirements. The memory requirement in CNNs can be viewed either as runtime CPU/GPU memory usage or storage space for the model. A number of recent works [7, 1, 53, 54, 49, 50, 64, 14, 17, 48, 33, 55, 52, 51, 31, 25] have explored such possibilities for efficient deep learning.

Most existing model compression/pruning methods can be divided into three categories. The first category [5, 11] has broadly considered introducing sparsity into the model parameters. These approaches give a limited compression rate on FLOPs and Total Runtime Memory (TRM), and need special software (sparse libraries) support to get the desired compression. These approaches provide a good compression rate in terms of weights storage, but provide limited FLOPs and TRM compression.

The second category of methods [11, 41, 30, 39, 35] has broadly focused on quantization based pruning. Special hardware is needed to provide the acceleration for the final compressed model. These kinds of model compression techniques are primarily designed for IoT devices.

The third category of methods [7, 1, 64, 50, 49] focuses on filter pruning. These approaches are generic and can be used practically without needing any special software/hardware needed for acceleration. These approaches provide a high compression rate in terms of FLOPs and TRM because of pruning of the whole convolutional filters from the model, which also reduces the depth of the feature maps. Sparsity and quantization based approaches are complementary to these approaches.

Filter pruning approaches require some measure to calculate the importance of the filter, which is a difficult task in general, and many heuristics have been used to measure filter importance. For example, [1] used a brute-force approach to discard the filters. They prune each filter sequentially and measure the importance of filters based on their corresponding accuracy drop, which can be impractical for large networks. [25] uses the ℓ_1 norm (sum of absolute values) of the filter to measure the filter importance, assuming that a high ℓ_1 norm filter is more likely to have a bigger influence on the feature map. [37, 46] use Taylor expansion based filter importance, which is motivated by the early work on optimal brain damage [24, 12].

As discussed in [16], filter importance based pruning methods have certain limitations in the form of requirements that are not always met. Compressed models produced by such methods suffer from redundancy because these methods don't consider filter redundancy while pruning. Therefore, the filter importance based pruning method is unable to reduce the redundancy present in the model and achieve the optimal solution.

*Equal contribution.

In this work, we propose iteratively removing the filters containing redundant information and getting a subset of filters that are minimally correlated. As also evidenced in other recent work, uncorrelated filters help to reduce overfitting [6, 44] and give a compact model with minimal redundancy. There are no benefits by keeping redundant filters in the model, and it will only result in overfitting or reduced generalization performance [6, 44]. The works in [3, 58] also eliminate the redundancy in convolutional filters by applying clustering to feature maps to perform filter pruning. In contrast, we use filter correlation to measure the redundancy present in the pairs of filters.

In particular, we present a filter pruning approach based on the correlation coefficient (Pearson correlation coefficient) of filter pairs. Unlike other prior works [14, 17, 31, 25] on filter pruning, instead of measuring *individual* filter importance, we measure importance for each *pair* of filters. Filter pairs that have the largest correlation (high redundancy) are given the lowest importance and are chosen for further optimization, before eventual pruning. In the optimization step, we further increase the correlation between each chosen filter pair and then finally prune (discard) one filter from the pair. This optimization step helps to transfer the knowledge of the filter to another one before discarding it. For example, suppose two filters f_1, f_2 in a pair have 60% pairwise correlation. If we discard one filter, we lose some of the information. However, suppose prior to pruning, we optimize the model in a way such that the filter pair's correlation increases to 99%. If we now discard one of the filters, we lose little information, and it would be safe to prune one of the filters from the pair (since the two filters in the pair have a high degree of similarity), and finetuning can quickly recover it. Our approach starts with the pre-trained model and then iteratively prunes the redundant filters as shown in Figure 1.

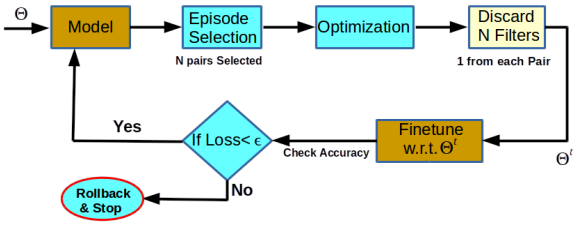

Figure 1. Correlation-based filter pruning where strongly correlated filter pairs are selected for optimization. The compression of the model depends on the user-defined error tolerance limit (ϵ). Hence ϵ can be seen as the stopping criteria in our approach.

2. Related Work

Most of the recent work on deep model compression can be categorized into three broad categories.

2.1. Connection Pruning

Connection pruning is a direct way to introduce sparsity into the CNN model. One approach for CNN compression is to prune the unimportant parameters. However, it is challenging to define the importance of parameters quantitatively. There are several approaches to measure the importance of the parameters. Optimal Brain Damage [24] and Optimal Brain Surgeon [12] used the second order Taylor expansion to calculate the parameters importance. However, the second order derivative calculations are very costly. [5] used hashing to randomly group the connection weights into a single bucket and then finetune the network. [62] proposed the skip layer approach for network compression. [11] proposed an iterative approach where absolute values of weights below a certain threshold are set to zero, and the drop in accuracy is recovered by finetuning. This approach is very successful when most of the parameters lie in the fully connected layer. The main limitation of these approaches is the requirement of special hardware/software for acceleration at run-time.

2.2. Filter Pruning

Filter pruning approaches (which is the focus of our work too) do not need any special hardware or software for acceleration. The basic idea in filter pruning [14, 17, 31, 25] is to get an estimate of the importance of the filters and discard the unimportant ones. After that, at each pruning step, re-training is needed to recover from the accuracy drop. [19] evaluates the importance of filter on a subset of the training data based on the output feature map. [1] used a greedy approach for pruning. They evaluated the filter importance by checking the model accuracy after pruning the filter. [37, 25] used similar approach but different measure for filter pruning. [7, 64, 20] used the low-rank approximation. [49] used a data-driven approach for calculating filter importance and pruning. [29] performed the channel level pruning based on the scaling factor in the training process. Recently, group sparsity is also a popular method for filter pruning. [50, 22, 61, 65, 2] explored the filter pruning based on the group sparsity.

2.3. Quantization

Weight quantization based approaches have also been used in prior works on model compression. [11, 9, 56] compressed CNN by combining pruning, quantization, and huffman coding. [34] conducted the network compression based on the float value quantization for model storage. Binarization [41] can be used for the network compression where each float value is quantized to a binary value. Bayesian methods [30] are also used for the network quantization. The quantization methods require special hardware support to get the advantage of the compression.

Our method is generic and does not require any special hardware/software, with the special support we can further

increase the FLOPs and memory compression because our method is complementary to other pruning methods such as binary/quantized weights, connection pruning, etc.

3. Proposed Approach

3.1. Terminology

Let $\mathcal{L}_i$ be the i^{th} layer and $i \in [1, 2, \dots K]$ where K is the number of convolutional layers. The layer $\mathcal{L}_i$ has c_o filters which is the number of output channels. The set of filters at layer $\mathcal{L}_i$ is denoted as $\mathcal{F}_{\mathcal{L}_i}$, where $\mathcal{F}_{\mathcal{L}_i} = \{f_1, f_2, \dots, f_{c_o}\}$. Each filter f_i is of dimension (h_k, w_k, c_{in}), where h_k, w_k and c_{in} are height, width and number of input channels, respectively.

3.2. FLOPs and Memory Size Requirements

Here we provide a brief analysis to help illustrate the effect of architecture hyperparameters on the FLOPs and memory consumption (which we will use in our experimental results to compare the various approaches).

For a CNN model, the total number of FLOPs on the layer $\mathcal{L}_i$ (convolutional $(FLOPs_{conv})$ or fully connected $(FLOPs_{fc})$) with the batch size B can be given as:

$$FLOPs_{conv_i} = c_{in} w_k h_k w_o h_o c_o * B \tag{1}$$

$$FLOPs_{fc_i} = c_{in} c_o * B \tag{2}$$

Here (w_{in}, h_{in}, c_{in}) is the input feature map, (w_k, h_k, c_{in}) is the convolutional filter and (w_o, h_o, c_o) is the output feature map. The total FLOPs across network can be defined as:

$$FLOPs = \sum_{i=1}^{K} FLOPs_{conv_i} + \sum_{j=1}^{N} FLOPs_{fc_j} \tag{3}$$

Here K, N is the number of convolutional and fully connected layers respectively. Convolutional filter of dimension (w_k, h_k, c_{in}) is applied $w_o h_o c_o$ times to the input feature map of dimension (w_{in}, h_{in}, c_{in}) to produce output feature map of dimension (w_o, h_o, c_o). Also, there are two sources of memory consumption: 1- feature map size, 2- parameter weight size. There are some other memory consumptions as well, but these are not feasible to estimate like those that are related to the implementation details of the model, the framework used, etc. So we can estimate the lower bound of memory size. The estimated memory requirement for layer $\mathcal{L}_i$ can be calculated as

$$M_{fm_i} = 4 w_o h_o c_o * B \tag{4}$$

$$M_{w_i} = 4 w_k h_k c_{in} c_o \tag{5}$$

Where M_{fm_i} is memory required for the feature map (w_o, h_o, c_o) and M_{w_i} is the memory required for parameter storage at layer $\mathcal{L}_i$. So the total memory requirement across each layers can be calculated as:

$$TRM = \sum_{i=1}^{K+N} M_{fm_i} + \sum_{j=1}^{K+N} M_{w_j} \tag{6}$$

For the fully connected layer $w_k, h_k, w_o, h_o = 1$ and c_{in}, c_o are the number of incoming and outgoing connections respectively. For the convolutional layer c_{in}, c_o is the number of input and output channel respectively. Please note that M_{fm_i} depends on the batch size. Therefore the methods that are based on sparsity but not filter pruning only reduce M_{w_i}. For such approaches, the feature map size remains the same and grows linearly with respect to the batch size.

3.3. Our Approach

Our approach iterative prunes a pre-trained model. In each iteration, we choose correlated filter pairs from each layer and optimize the selected filter pairs such that filters in each selected pair are as highly correlated as possible. This enables us to safely discard, without information loss, one of the filters (which is redundant) from each of the selected pairs. We finetune the compressed model after pruning. This constitutes one iteration of pruning (Figure 1), which can be further repeated to more iterations until we could recover accuracy with in the tolerance limit.

In our approach, we consider the importance of filter *pairs* for pruning. We use the correlation coefficient of the filter pair to quantify their importance. The filter pair that has the largest correlation coefficient is defined as the *least important*. It is considered as the least important filter pair because of the presence of redundancy (one of the filters from the pair is redundant). However, if we drop one of the filters from this pair, we might also be losing their mutually-complementary information, which may or may not be captured by the other filters. Therefore, before discarding one of the filters from the pair, we need to transfer this information to the other filters by optimizing the model before pruning. There may be the case when a filter belongs to multiple highly correlated pairs, then that filter is pruned in the process. It may also be possible that both filters are important in highly correlated filter pair, and accuracy may drop after pruning, but the small finetuning (1-2 epoch) will recover it as we have only removed a redundant filter from the pair.

Some of the previous approaches like [8, 11, 25] used the ℓ_1 or ℓ_2 regularizer for defining the filter importance. However, using the ℓ_1 or ℓ_2 regularizer typically works well only when there is significant redundancy in the model. These approaches are unable to give a highly compressed model without sacrificing accuracy. If we try to get a highly compact model with these approaches, the system performance degrades rapidly because the strong ℓ_1 or ℓ_2 regularizer

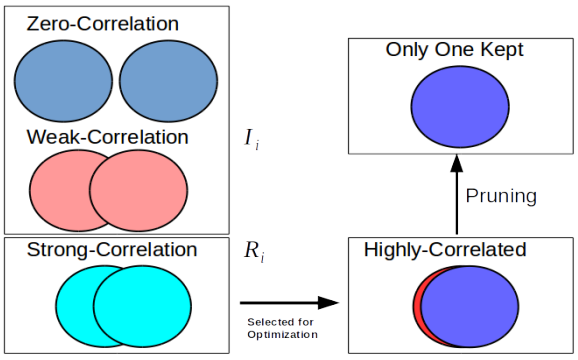

Figure 2. Strongly correlated filters are selected and make them highly correlated. Finally, one redundant filter is pruned.

starts penalizing the important filters as well and, therefore, cannot achieve a significant compression. If we increase the ℓ_2 penalty, it starts controlling the model complexity. Large ℓ_2 penalty results in underfitting. Therefore these approaches can only lead to limited levels of filter pruning. In contrast, in our proposed approach, we learn a minimal basis of filters that are highly uncorrelated. Figure 2 illustrates the basic idea of our approach.

Correlation is the most common and one of the simplest statistical measures to quantify data dependence or association. Correlation often refers to how close two variables are to have a linear relationship with each other. Let $\mathbf{X}$ and $\mathbf{Y}$ are two random variables with expected values μ_X and μ_Y and standard deviations σ_X and σ_Y respectively. The correlation ρ_{XY} can be defined as

$$\rho_{XY} = \frac{\mathbb{E}[(\mathbf{X} - \mu_X)(\mathbf{Y} - \mu_Y)]}{\sigma_X \sigma_Y} \tag{7}$$

Here $\rho_{XY} \in [-1, 1]$, with a high negative or high positive value indicates a high degree of linear dependence between the two variables. When the correlation is near zero, the two variables are linearly independent.

3.3.1 Episode Selection

Let us define the two terms which we will use in the proposed pruning process - "Ready-to-prune ($\mathbf{R_i}$)" and "Pruned ($\mathbf{P_i}$)". Here $\mathbf{R_i}$ denotes filter pairs selected from the layer $\mathcal{L}_i$ for the optimization process (details in the next section). $\mathbf{P_i}$ denotes the filters that are eventually selected (one from each pair of $\mathbf{R_i}$) for pruning from the model after optimization. Therefore if N filter pairs are selected in $\mathbf{R_i}$ then $|\mathbf{P_i}| = N$, i.e., from the layer $\mathcal{L}_i$, N filters will get pruned.

In each layer $\mathcal{L}_i$, we find out the filter pairs that have the maximum correlation. For calculating the correlation, we select all filters $\mathcal{F}_{\mathcal{L}_i} = \{f_1, f_2, \ldots, f_{c_o}\}$ on a layer $\mathcal{L}_i$ with c_o output channels. Each filter f_i on the layer $\mathcal{L}_i$ is of dimension (w_k, h_k, c_{in}) is flattened to a vector of size $w_k \times h_k \times c_{in}$. Now we can calculate the correlation coefficient for filter pair by using Eq. 7. In our approach, we have considered the magnitude of the correlation coefficient, thereby giving the same importance for positive and negative correlation values. Based on the magnitude of the correlation coefficient of each pair, filter pairs are ordered.

The filter pair with the largest correlation value has the minimum importance. Some least important filter pairs are selected for the ready-to-prune set $\mathbf{R_i}$ (select the filter pair (f_a, f_b) such that $a \neq b$). Let $\mathbf{I_i}$ are the remaining filter pairs at layer $\mathcal{L}_i$. Then:

$$\mathcal{PF}_{\mathcal{L}_i} = \mathbf{R_i} \cup \mathbf{I_i} \quad s.t. \quad \mathbf{R_i} \cap \mathbf{I_i} = \phi \tag{8}$$

Here $\mathcal{PF}_{\mathcal{L}_i} = \{(f_a, f_b) : a \in \{1, 2, \ldots c_o\}, \quad b \in \{1, 2, \ldots c_o\}\}$. The same process is repeated for each layer. This selected set of filter pairs from all the layers ($\mathbf{S_t}$) is called one episode. Where $\mathbf{S_t}$ is the set of filter pairs selected at t^{th} episode from all K convolutional layers.

$$\mathbf{S_t} = \{\mathbf{R_1}, \mathbf{R_2}, \ldots, \mathbf{R_K}\} \tag{9}$$

This set $\mathbf{S_t}$ is the collection of the ready-to-prune filter pairs from all the layers and used for further optimization such that both filters in each pair contain similar information after optimization. Therefore we can safely remove one filter from each pair. The optimization process is explained in the next section.

3.3.2 Optimization

Let $C(\Theta)$ be the cost function of the original model and Θ be the model parameters. We optimize this cost function with the new regularizer applied to the selected episode (set of filter pairs) $\mathbf{S_t}$. Our new regularizer $(C_{\mathbf{S_t}})$ is as follows:

$$C_{\mathbf{S_t}} = \exp\left(-\sum_{X,Y \in \mathbf{R_i}, \forall \mathbf{R_i} \in \mathbf{S_t}} |\rho_{XY}|\right) \tag{10}$$

Here $|\rho_{XY}|$ is the magnitude of correlation coefficient of the filter pair (X, Y) in $\mathbf{R_i}$ and $\mathbf{R_i} \in \mathbf{S_t}$. The idea is here to make the strongly correlated filter pair highly correlated (as illustrated in Figure 2). Note that Eq. 10 will be minimized when $\sum_{X,Y \in \mathbf{R_i}, \forall \mathbf{R_i} \in \mathbf{S_t}} |\rho_{XY}|$ term is maximum, i.e. each filter pair's magnitude of correlation coefficient $|\rho_{XY}| \to 1$. This new regularizer is added to the original cost function so our new objective function is as follows:

$$\Theta = \arg\min_{\Theta} \left(C(\Theta) + \lambda * C_{\mathbf{S_t}}\right) \tag{11}$$

Here λ is the hyper-parameter to control the regularization term. Minimizing the Eq. 11 will optimize such that without degrading the model performance, it increases the correlation as high as possible between filters in each pair belonging to $\mathbf{R_i}$. Therefore we can safely remove one filter from each pair.

3.3.3 Pruning and Finetuning

After increasing the correlation between filters in each pair belonging to $\mathbf{R_i}$, we can prune one filter from each pair belonging to $\mathbf{R_i}$. Our model has a reduced set of the parameter Θ'

$$\Theta' = \Theta \setminus \{\mathbf{p_1}, \mathbf{p_2} \cdots \mathbf{p_k}\} \tag{12}$$

Where $\mathbf{p_i}$ is the set of filters finally selected to be removed from the model.

Further, we finetune the model w.r.t. the parameter Θ'. Since we discarded the *redundant* filters from the model, the information loss from the model would be minimum. Hence the finetuning process can easily recover from the small loss, and the model's performance can be brought back to be nearly the same as the original one. Please note that, if two filters are highly correlated, then their output feature maps are also highly correlated because both filters are applied to the same input feature maps.

4. Experiments

To evaluate our approach Correlated Filter Pruning (CFP), we use three standard models, LeNet-5 [23], VGG-16 [47] and ResNet-50,56 [13], for classification, and two popular models, Faster-RCNN and SSD, for object detection. Our experiments were done on GTX 1080 Ti GPU and i7-4770 CPU@3.40GHz. Through an extensive set of experiments, we show that our proposed approach achieves state-of-art compression results. In all our experiments, we set $\lambda = 1$ to enforce a high correlation for optimizing Eq. 11. We can also choose a smaller value of λ, but then it would increase the number of epochs in the optimization step. A very high value of λ may result in accuracy loss because now optimization gives more weight to $C_{\mathbf{S_t}}$ (Eq. 11) than $C(\Theta)$. We empirically found $\lambda = 1$ is the right choice for all our experiments. Filter pairs selection for the optimization process ($\mathbf{R_i}$) is proportional to the FLOPs on layer $\mathcal{L}_i$ to reduce the same % of FLOPs from every layer. We simultaneously prune filters across all layers. We continued this filter pairs selection strategy until we can recover the accuracy with in the tolerance limit. Ones the tolerance limit is achieved, we start individual layer pruning. We pruned each layer sequentially from start to end one by one until we could recover accuracy with in the tolerance limit.

4.1. LeNet-5 on MNIST

MNIST is a handwritten digit dataset contains 60,000 images for training and 10,000 images for the testing. We use the LeNet-5 architecture that contains two convolutional layers and two fully connected layers. The complete architecture is 20-50-800-500, where 20, 50 are the number of convolutional filters in first and second convolutional layers respectively. We trained the model from scratch and achieved an error rate of 0.83%.

Method	r_1, r_2	Error%	FLOPs	Pruned Flop %
SSL-2 [61]	5,19	0.80	5.97×10^5	86.42
SSL-3 [61]	3,12	1.00	2.89×10^5	93.42
SBP [38]	–	0.86	–	90.47
SparseVD [36]	–	0.75	–	54.34
Baseline	20,50	0.83 ± 0.09	4.40×10^6	0.0
CFP-1 (Ours)	**4,5**	$\mathbf{0.91 \pm 0.07}$	$\mathbf{1.95 \times 10^5}$	**95.56**
CFP-2 (Ours)	**3,5**	$\mathbf{0.95 \pm 0.05}$	$\mathbf{1.58 \times 10^5}$	**96.41**
CFP-3 (Ours)	**3,4**	$\mathbf{1.20 \pm 0.09}$	$\mathbf{1.39 \times 10^5}$	**96.84**
CFP-4 (Ours)	**2,3**	$\mathbf{1.77 \pm 0.08}$	$\mathbf{0.89 \times 10^5}$	**97.98**

Table 1. Pruning results for the LeNet-5 architecture on MNIST. Here r_1, r_2 are used to denote the number of remaining filters in first and second convolutional layers respectively. We run each experiment three times and report the "mean±std".

To show the effectiveness of our proposed approach, we conduct the first experiment on LeNet-5 for the MNIST dataset. As compared to the previous approaches, we achieve a much higher FLOPs compression rate with a relatively small accuracy drop. In prior work, SSL-3 [61], report an error of 1.0% on 93.42% FLOPs pruning while we achieve 96.41% FLOPs pruning with only 0.95% error. Also, note that, as compared to SBP [38], that has only 90.47% FLOPs pruning with 0.86% error, our approach has 95.56% FLOPs pruning with the negligible (0.05%) error difference. Please refer to Table 1 for a detailed comparison. CFP-1 denotes the first compressed model in this iterative pruning scheme when it shows a competitive accuracy as compared to other approaches. We can repeat the process for more iterations to compress the model further and can monitor the accuracy after each pruning iteration. This helps us assess our approach's ability to compress the model further. CPF-2,3 and 4 denote the compressed models obtained by such iterative pruning after 2, 3, and 4 iterations, respectively.

4.2. VGG-16 on CIFAR-10

We next experiment with the VGG-16 architecture on CIFAR-10. Each image size is of size 32×32 on the RGB scale. The VGG-16 convolutional layers architecture is the same as [47], and after each convolutional layer, batch normalization layers are added. We use the same architecture and settings as described in [25]. The network is trained from scratch and achieves a 6.51% error rate. Figure 3 shows the layer-wise FLOPs distribution for the original and pruned model.

Like the LeNet-5 pruning results, we observe the same pattern for the VGG-16 pruning on the CIFAR-10 dataset. We have 80.36% FLOPs pruning with a 6.77% error while previous state-of-art approach SBPa [38] has only 68.35% FLOPs pruning with 9.0% error. SparseVD [36] has 55.95% pruning with 7.2% error, while we have 81.93% pruning with significantly less (7.02%) error. Please refer to Table 2 for detail comparison. In the Table 2, CFP-1, and

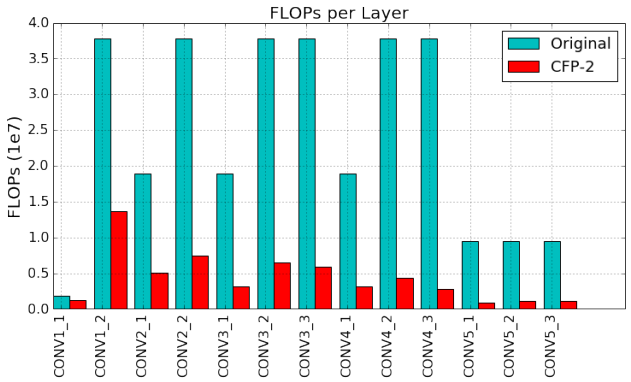

Figure 3. The original and pruned model FLOPs on each layer for VGG-16 on CIFAR-10.

Method	Error%	FLOPs	Pruned Flop%
Li-pruned [25]	6.60	2.06×10^8	34.20
SBPa [38]	9.00	–	68.35
SBP [38]	7.50	–	56.52
SparseVD [36]	7.20	–	55.95
Baseline	6.51 ± 0.23	3.137×10^8	0.0
CFP-1 (Ours)	$\mathbf{6.77 \pm 0.19}$	$\mathbf{6.16 \times 10^7}$	**80.36**
CFP-2 (Ours)	$\mathbf{7.02 \pm 0.16}$	$\mathbf{5.67 \times 10^7}$	**81.93**

Table 2. Pruning results for VGG-16 architecture on the CIFAR-10 dataset. We run each experiment three times and report the "mean±std".

CPF-2 denote the first and second compressed models respectively (in this iterative pruning scheme). The original and pruned FLOPs are shown in Figure 3.

4.3. Results on ResNet

4.3.1 ResNet-56 on CIFAR-10

We use the ResNet-56 architecture [13] on the CIFAR-10 dataset, which contains the three stages of the convolutional layer of size 16-32-64, where each convolution layer in each stage contains the same 2.36M FLOPs. We trained the model from scratch using the same parameters proposed by [13] and achieve the error rate of 6.43%. The network is pruned into two cycles. In the initial cycle, we selected 1 filter pair from each convolutional layer in the first stage (total 9 filter pairs), 2 filter pairs from each convolutional layer in the second stage (total 18 filter pairs) and 4 filter pairs from each convolutional layer in the third stage (total 36 filter pairs) to prune the same amount of FLOPs from each stage for $\mathbf{S_t}$ and pruned one filter from each pair (total 9, 18 and 36 filters pruned from first, second and third stage respectively), till we can recover the accuracy drop with the ϵ tolerance. In the second cycle, we selected one filter pair (total nine filter pairs from all convolutional layers in a particular stage) in a particular stage for the $\mathbf{S_t}$ and pruning continue till we can recover the model accuracy with the ϵ tolerance. The results are shown in Table 3. It is clear

Method	r_1, r_2, r_3	Error%	FLOPs	Pruned Flop %
Li-A [25]	—	6.90	1.12×10^8	10.40
Li-B [25]	—	6.94	9.04×10^7	27.60
NISP [63]	—	6.99	–	43.61
CP [17]	—	8.20	–	50.00
SFP [14]	—	6.65	–	52.60
AMC [15]	—	8.10	–	50.00
Baseline	16,32,64	6.43 ± 0.15	1.26×10^8	0.0
CFP-1	10,20,38	$\mathbf{6.68 \pm 0.12}$	$\mathbf{4.85 \times 10^7}$	**61.51**
CFP-2	9,18,36	$\mathbf{6.93 \pm 0.10}$	$\mathbf{4.08 \times 10^7}$	**67.62**
CFP-3	8,16,27	$\mathbf{7.37 \pm 0.17}$	$\mathbf{2.95 \times 10^7}$	**76.59**

Table 3. Pruning results for ResNet-56 architecture on CIFAR-10. We run each experiment three times and report the "mean±std".

from the table that as compared to the previous approach, our method shows the state of art result. CP [17] has 50.0% FLOPs pruning with a 8.2% error, while with only 7.37% error, we have significantly higher 76.56% FLOPs pruning. Similarly, SFP [14] has 52.6% pruning with a 6.65% error, and our model has 61.51% pruning with the same error. Here r_1, r_2, r_3 are used to denote the number of remaining filters in each convolutional layer within the three stages. We use the same approach as [8] to resolve the skip connection inconsistency during the filter pruning.

4.3.2 ResNet-50 on ImageNet

We experiment with the ResNet-50 model on the large-scale ImageNet [45] dataset. The results are shown in Table 4. We follow the same settings as mentioned in [17]. As compared to ThiNet-50, we have similar FLOPs pruning with significantly better accuracy.

Other proposed approaches, such as channel pruning (CP) [17] and structured probabilistic pruning (SPP) [59] have $\sim$ 50% FLOPs pruning, but their error rate is high. In particular, CP has a 9.2% error, and SPP has a 9.6% error. Our proposed approach is highly competitive with these approaches in terms of FLOPs pruning, while also yielding significantly better accuracy. Please refer to Table 4 for more details.

4.4. Ablation Study

In the following section, we present an ablation study on how our correlation-based criterion helps in preserving the information in the model during filter pruning, how finetuning helps after discarding one of the filters from the filter pair and analyze the correlations among the filters retained in the final model.

4.4.1 Optimization w.r.t. Accuracy

Recall that, before discarding filters directly based on correlation, we further optimize the filter correlation such that the

Method	Error (%)	Pruned Flop (%)
ThiNet-50 [32]	9.0	~ 50
CP [17]	9.2	~ 50
SPP [59]	9.6	~ 50
WAE [4]	9.6	46.8
Baseline	7.8	0.0
CFP (Ours)	**8.6**	**49.6**

Table 4. ResNet-50 Pruning results on the ImageNet with the other state-of-art approaches. The baseline network's top-5 accuracy is 92.2% (https://github.com/KaimingHe/deep-residual-networks).

Model	OPT	Error (%)	PF (%)
LeNet-5 (CFP-3)	No	1.81	96.84
	Yes	**1.20**	96.84
LeNet-5 (CFP-4)	No	2.32	97.98
	Yes	**1.77**	97.98
VGG-16 (CFP-1)	No	7.20	80.36
	Yes	**6.77**	80.36
VGG-16 (CFP-2)	No	7.61	81.93
	Yes	**7.02**	81.93

Table 5. Effect of the correlation optimization given by equation-[10] for the LeNet-5 and VGG-16 (**OPT**: Optimization, and **PF**: Pruned Flop).

strongly correlated filter becomes even more strongly correlated (using the regularizer based on Eq. 10). We have found that if we discard the filter without optimization, the model suffers from the significant accuracy drop due to the loss of potentially mutually-complementary information. Therefore, before discarding one of the filters from the pair, we need to transfer this information to the other filters that remain in the model after pruning using Eq. 11.

Please refer to Table 5 to see the effect of the optimization. LeNet and VGG-16 compressed models achieve better accuracy with optimization.

4.4.2 Analysis of correlation among filters in the final compressed model

In the case of VGG-16 on the CIFAR-10, initially, the maximum filter correlation is 0.7, but in the final compressed model, the maximum filter correlation is nearly 0.1, which shows that we have successfully removed the redundant filters. At the same time, the fact that the classification accuracy does not drop much (Table 2), indicates that the useful discriminative filters are preserved. Here, we find a subset of filters that are minimally correlated but preserves maximal information for our finally compressed model. These uncorrelated filters help to reduce the overfitting [6, 44] and give a compact model with the least possible redundancy.

4.5. Speedup and Memory Size

The pruned FLOPs is not necessarily equivalent to practical model speedup because FLOPs give the theoretical

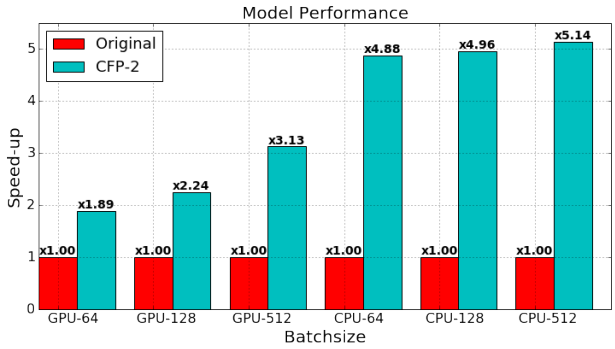

Figure 4. Model performance on the CPU (i7-4770 CPU@3.40GHz) and GPU (TITAN GTX-1080 Ti) for VGG-16 on CIFAR-10.

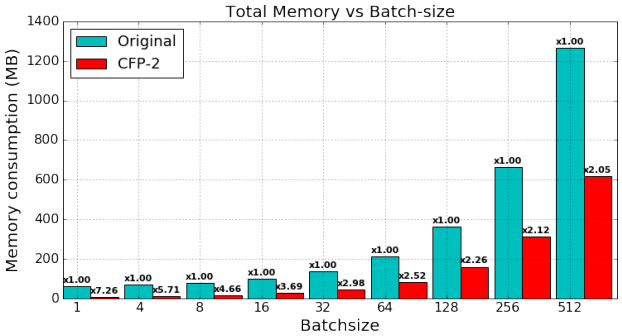

Figure 5. Increase in the total memory size w.r.t. batch size for VGG-16 on CIFAR-10.

speedup. The practical speedup can be very different from the result reported in terms of pruned FLOPs percentage. The practical speedup depends on the many other factors, for example, parallelization bottleneck on intermediate layers, I/O operation, etc. Also, total run-time memory (TRM) does not depend only on the compressed model parameters size but also on the feature maps (FM), batch-size (BS), the dynamic library used by Cuda, all the supporting header-file, etc. Here we don't have control over all the parameters but Model parameters size (MPS), FM, and BS. To show the practical speedup and Memory size, we experiment with the VGG-16 model over the CIFAR-10 dataset. The result for the speedup and TRM are shown in the Figure 4 and Figure 5 respectively.

FM is the most important factor for reducing the run-time memory since it grows linearly w.r.t. batch size and quadratic w.r.t. image size while MPS is fixed. The filter pruning approach reduces the model parameters as well as feature maps memory size while all the approaches based on sparsity in the model are reducing only the MPS, and the size of the FM remains the same. Hence batch size has the bottleneck. If we have a limited batch size, this reduces the parallelism in the GPU, resulting in the speed drop. TRM can be defined as:

$$TRM = MPS + (FM * 4 * BS) \tag{13}$$

Model	AP				Size	Parameters
	prohibitory	mandatory	danger	mAP		
SSD512-O	96.83	86.93	87.05	90.27	98.7 MB	24.7M
SSD512-P	98.35	88.45	89.01	**91.94**	**4.0 MB (24.7×)**	**0.99M (4.0%)**

Table 6. Class wise AP for SSD512-original(O) and SSD512-pruned(P) model on GTSDB dataset.

Model	data	Avg. Precision, IoU:			Avg. Precision, Area			Avg. Recall, #Dets:			Avg. Recall, Area:		
		0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
F-RCNN original	trainval35K	27.5	48.7	28.1	12.4	31.5	38.7	25.7	38.8	39.8	21.2	45.5	55.0
F-RCNN pruned	trainval35K	27.8	48.4	28.5	13.3	31.6	38.6	26.1	39.5	40.6	22.6	45.7	55.4

Table 7. Generalization results are shown on the MS-COCO dataset. Pruned ResNet-50 (CFP) used as a base model for Faster-RCNN. Where S, M, and L are the small, medium, and large area ranges respectively for evaluation. 1, 10, and 100 denotes thresholds on max detections per image (http://cocodataset.org/#detection-eval).

It is clear from Figure 5 that with the increase in BS, TRM memory increases. Therefore we cannot go for a big batch size. While Figure 4 shows that for the small batch size system performance (speedup) degraded. Therefore for the speedup, we have to choose a bigger BS, but there is a memory bottleneck on the GPU or CPU. Hence in the proposed approach, we prune the whole convolutional filter so that FM memory can be reduced.

4.6. Generalization Ability

4.6.1 Compression for Object Detection

To show the generalization ability of our approach, we also show the result on the detection network. In this experiment we have taken two most popular object detector SSD [28] on GTSDB dataset and Faster RCNN [43] on MS-COCO [27]. In the case of SSD, we achieve $\sim$25$\times$ compression in terms of model parameters with significant improvement in AP. For faster RCNN, we have used ResNet-50 as a base network.

4.6.2 SSD512 on German traffic detection benchmarks

In this experiment, we evaluate the generalization ability of our pruned model, VGG-16 CFP-2, which is pruned on CIFAR-10. First, we trained original SSD512 on German traffic detection benchmarks (GTSDB) [18] dataset, In which ImageNet pre-trained base network was used. In the second case, we replace the base network of SSD512 with our pruned model, VGG-16 CFP-2. After training, we analyzed the model and found the model is over-fitted because GTSDB is a small scale dataset. Our pruned SSD512 model detects the object from the initial layer only, which is CONV4_3, and we removed all other remaining layers after CONV4_3. After doing this, we observed a significant improvement in the mAP and $\sim$25$\times$ compression in model size. Hence our pruned model successfully generalizes the object detection task on the GTSDB dataset. Refer to Table 6 for the detailed experimental results.

4.6.3 Faster RCNN on COCO

We experiment on the COCO detection datasets with 80 object classes [27]. 80k train images and 35k val images are used for training (trainval35K) [26]. We report the detection accuracies over the 5k unused val images (minival). In this first, we trained Faster-RCNN with the ImageNet pre-trained ResNet-50 base model. The results are shown in Table 7. In the second experiment, we used our pruned ResNet-50 model (CFP), which is pruned on the ILSVRC-2012 dataset as given in Table 4. Then we used our pruned ResNet-50 (CFP) model as a base network in Faster-RCNN. In the Faster-RCNN implementation, we used ROI Align instead of ROI Pooling. We found that the pruned model shows slightly better performances in some cases (mAP@0.75, mAP@0.5:0.95). Refer to the Table 7 for more details.

5. Conclusion

We have proposed a novel approach for filter pruning, which is guided by pairwise correlations of filters. Unlike the previous heuristics for measuring individual filters importance for pruning, we proposed a new approach for considering filter pairs importance based on the redundancy present in the pair. In the pruning process, we iteratively reduce the redundancy in the model. Our approach, as compared to the existing methods, shows state-of-art results. The efficacy of our method is demonstrated via a comprehensive set of experiments and ablation studies. We have shown the generalization capability of our approach for the object detection task.

Acknowledgment:

PS is supported by the Research-I Foundation at IIT Kanpur. VKV acknowledges support from Visvesvaraya PhD Fellowship and PR acknowledges support from Visvesvaraya Young Faculty Fellowship.

References

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

[2] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *NIPS*, pages 2270–2278, 2016.

[3] B. O. Ayinde and J. M. Zurada. Building efficient convnets using redundant feature pruning. *ICLR*, 2018.

[4] T. Chen, L. Lin, W. Zuo, X. Luo, and L. Zhang. Learning a wavelet-like auto-encoder to accelerate deep neural networks. *AAAI*, 2018.

[5] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *ICML*, pages 2285–2294, 2015.

[6] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra. Reducing overfitting in deep networks by decorrelating representations. *ICLR*, 2016.

[7] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

[8] X. Ding, G. Ding, J. Han, and S. Tang. Auto-balanced filter pruning for efficient convolutional neural networks. *AAAI*, 2018.

[9] A. Dubey, M. Chatterjee, and N. Ahuja. Coreset-based neural network compression. 2018.

[10] C. Finn, K. Xu, and S. Levine. Probabilistic model-agnostic meta-learning. *NIPS*, 2018.

[11] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[12] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*, 1993.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[14] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks. *IJCAI*, 2018.

[15] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[16] Y. He, P. Liu, Z. Wang, and Y. Yang. Pruning filter via geometric median for deep convolutional neural networks acceleration. *CVPR*, 2019.

[17] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, page 6, 2017.

[18] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *IJCNN*, number 1288, 2013.

[19] 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.

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

[21] T. Kim, J. Yoon, O. Dia, S. Kim, Y. Bengio, and S. Ahn. Bayesian model-agnostic meta-learning. *NIPS*, 2018.

[22] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In *CVPR*, pages 2554–2564, 2016.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *NIPS*, pages 598–605, 1990.

[25] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *ICLR*, 2017.

[26] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. In *CVPR*, page 4, 2017.

[27] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014.

[28] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *ECCV*, pages 21–37. Springer, 2016.

[29] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, pages 2755–2763. IEEE, 2017.

[30] C. Louizos, K. Ullrich, and M. Welling. Bayesian compression for deep learning. In *NIPS*, pages 3288–3298, 2017.

[31] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5058–5066, 2017.

[32] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, and W. Lin. Thinet: pruning cnn filters for a thinner net. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[33] P. Mazumder, P. Singh, and V. Namboodiri. Cpwc: Contextual point wise convolution for object recognition. *arXiv preprint arXiv:1910.09643*, 2019.

[34] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Towards unified data and lifecycle management for deep learning. In *ICDE*, pages 571–582. IEEE, 2017.

[35] A. Mishra and D. Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

[36] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. In *ICML*, pages 2498–2507, 2017.

[37] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *ICLR*, 2017.

[38] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *NIPS*, pages 6775–6784, 2017.

[39] A. Polino, R. Pascanu, and D. Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[40] S. Qiao, C. Liu, W. Shen, and A. Yuille. Few-shot image recognition by predicting parameters from activations. 2018.

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

[42] S. Reed, Y. Chen, T. Paine, A. van den Oord, S. M. A. Eslami, D. Rezende, O. Vinyals, and N. de Freitas. Few-shot autoregressive density estimation: Towards learning to learn distributions. In *ICLR*, 2018.

[43] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015.

[44] P. Rodríguez, J. Gonzàlez, G. Cucurull, J. M. Gonfaus, and X. Roca. Regularizing cnns with locally constrained decorrelations. *ICLR*, 2017.

[45] 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. *IJCV*, 115(3):211–252, 2015.

[46] A. Sharma, N. Wolfe, and B. Raj. The incredible shrinking neural network: New perspectives on learning representations through the lens of pruning. *arXiv preprint arXiv:1701.04465*, 2017.

[47] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[48] P. Singh, V. S. R. Kadi, and V. P. Namboodiri. Falf convnets: Fatuous auxiliary loss based filter-pruning for efficient deep cnns. *Image and Vision Computing*, page 103857, 2019.

[49] P. Singh, V. S. R. Kadi, N. Verma, and V. P. Namboodiri. Stability based filter pruning for accelerating deep cnns. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1166–1174. IEEE, 2019.

[50] P. Singh, R. Manikandan, N. Matiyali, and V. Namboodiri. Multi-layer pruning framework for compressing single shot multibox detector. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1318–1327. IEEE, 2019.

[51] P. Singh, P. Mazumder, and V. P. Namboodiri. Accuracy booster: Performance boosting using feature map recalibration. *arXiv preprint arXiv:1903.04407*, 2019.

[52] P. Singh, M. Varshney, and V. P. Namboodiri. Cooperative initialization based deep neural network training. *arXiv preprint arXiv:2001.01240*, 2020.

[53] P. Singh, V. K. Verma, P. Rai, and V. P. Namboodiri. Hetconv: Beyond homogeneous convolution kernels for deep cnns. *International Journal of Computer Vision*, pages 1–21, 2019.

[54] P. Singh, V. K. Verma, P. Rai, and V. P. Namboodiri. Hetconv: Heterogeneous kernel-based convolutions for deep cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4835–4844, 2019.

[55] P. Singh, V. K. Verma, P. Rai, and V. P. Namboodiri. Play and prune: Adaptive filter pruning for deep model compression. *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[56] F. Tung and G. Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[57] V. K. Verma, G. Arora, A. Mishra, and P. Rai. Generalized zero-shot learning via synthesized examples. *CVPR*, 2018.

[58] 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.

[59] H. Wang, Q. Zhang, Y. Wang, and H. Hu. Structured probabilistic pruning for convolutional neural network acceleration. *BMVC*, 2017.

[60] W. Wang, Y. Pu, V. K. Verma, K. Fan, Y. Zhang, C. Chen, P. Rai, and L. Carin. Zero-shot learning via class-conditioned deep generative models. *AAAI*, 2017.

[61] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *NIPS*, pages 2074–2082, 2016.

[62] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris. Blockdrop: Dynamic inference paths in residual networks. In *CVPR*, pages 8817–8826, 2018.

[63] 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. *CVPR*, 2018.

[64] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *NIPS*, pages 1984–1992, 2015.

[65] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: Towards compact cnns. In *ECCV*, pages 662–677. Springer, 2016.